HAL
open science

SVSP (Secure Voice over IP Simple Protocol) une solution pour la sécurisation de la voix sur IP

Carole Bassil

To cite this version:

Carole Bassil. SVSP (Secure Voice over IP Simple Protocol) une solution pour la sécurisation de la voix sur IP. domain_other. Télécom ParisTech, 2005. Français. NNT : . pastel-00001577

HAL Id: pastel-00001577

https://pastel.hal.science/pastel-00001577

Submitted on 22 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Travaux de recherche

Présentée pour l'obtention du titre de DOCTEUR de

L'Ecole Nationale Supérieure des Télécommunications

Discipline : Informatique et Réseaux

Par

Carole Bassil

SVSP (Secure Voice over IP Simple Protocol) Une solution pour la sécurisation de la voix sur IP

Soutenance le 12 décembre 2005 devant le jury composé de :

Jean-Pierre Claudé Professeur Université de Versailles Rapporteurs
Abdelmadjid Bouabdallah Professeur Université de Technologie de Compiègne

Ken Chen Professeur Université Paris 13 Examinateurs
Noémie Simoni Professeur ENST Paris

Ahmed Serhrouchni Maître de conférence ENST Paris Directeurs de Thèse
Nicolas Rouhana Professeur Université de Saint Joseph Beyrouth

A ma mère, mon père,
ma sœur et sa famille,
à Tony, l'amour de ma vie…

« Le maître dit :
Si vous suivez le chemin de vos rêves,
engagez-vous vraiment…
Envisagez votre chemin avec courage
et ne craignez pas les critiques d'autrui.
Surtout, ne vous laisser pas paralyser par l'autocritique.
Dieu sera avec vous durant vos nuits d'insomnie,
et son Amour séchera vos larmes secrètes.
Dieu est le Dieu des vaillants.»
Maktub – Paulo Coelho

Remerciements

Je tiens à remercier en tout premier lieu mon directeur de thèse, M. Ahmed Serhrouchni. Tout au long de ces quatre années, il a su orienter mes recherches aux bons moments, me guider par ses conseils judicieux et par ses idées innovantes. Je te remercie pour ta patience, pour avoir cru en moi et pour m'avoir encourager dans les moments difficiles. Je n'ai et n'aurai jamais assez de mots pour te remercier pour ton amitié qui m'est devenue très chère.

Je tiens à remercier aussi mon co-directeur de thèse, M. Nicolas Rouhana, pour l'aide si efficace qu'il m'a apportée. Sans son intervention, il m'aurait été très difficile de poursuivre ce projet. Je le remercie aussi pour sa patience, pour le temps précieux qu'il m'a accordé et surtout pour son amitié.

Je remercie MM. Jean-Pierre Claudé et Abdelmadjid Bouabdallah pour avoir accepté d'être les rapporteurs de cette thèse et pour la rapidité avec laquelle ils ont lu mon manuscrit et l'intérêt qu'ils ont porté à mon travail. Merci également aux autres membres du jury qui ont accepté de juger ce travail : Mme Noémie Simoni et M. Ken Chen.

Je remercie infiniment l'Agence Universitaire de Francophonie pour m'avoir accordée une bourse qui m'a permis de poursuivre mes recherches à l'ENST de Paris et pour m'avoir ouvert les portes de son Campus Numérique à Berytech pour y travailler. Je tiens à remercier spécifiquement l'équipe de l'AUF Mme Emmanuelle Nasser, Melle Mirande Sfeir, et l'acceuil chaleureux et l'amitié de Mme Mireille El Rayess.

Mes plus chaleureux remerciements s'adressent à mes collègues particulièrement M. Walid Badaoui pour nos discussions, ses conseils et son support technique ainsi qu'à mes étudiants pour leur enthousiasme, spécialement à Antoine Khalifé qui a participé activement à la partie implémentation de cette thèse.

Je voudrais également remercier mes amis, Toni, Nathalie, Salma, Amine, Spiro, Odile, Carla, Roula, Patrick, Shérine, Hassan, Jacques, Chadi, Marguerite… (Pardonner moi si j'ai oublié quelqu'un) pour leur support et leur soutien moral qu'ils m'auront fournis tout au long de la réalisation de ces travaux. Je n'oublierai pas aussi l'aide précieuse et les discussions sur ce projet avec Ibrahim que je remercie tout particulièrement. Une pensée spéciale est adressée à une personne très chère, M. Michel Abou Zeid, pour ses encouragements et pour sa confiance.

Mes pensées finales reviendront à ma mère, mon père et ma soeur qui m'ont soutenus et entourés par leur amour, leur tendresse, leur patience, leurs prières, pendant toutes ces années, chacun à sa manière, et sans qui je n'aurais pas pu aller au bout de mon projet…, ainsi qu'à l'ensemble de ma famille, surtout mon neveu Joey qui par ses mimiques et son air angélique m'a apporté le sourire… Merci.

Cette thèse t'est dédiée, à toi, Tony, qui a toujours été présent à mes cotés pour me soutenir, m'encourager…Je te remercie du fond de mon coeur pour tout. Mais, tu le sais, le plus beau est à venir pour nous deux ! Il nous reste la vie entière pour en profiter. Merci mon Tantouneh…

Enfin, je n'en finirai pas de remercier Dieu, la Sainte Trinité et la Sainte Vierge qui, sans Eux, je ne serais jamais ce que je suis…

Résumé

Depuis l'invention du premier téléphone par Alexandre Graham Bell en 1869, la téléphonie n'a cessé d'évoluer : de la commutation de circuit à la commutation par paquet, d'un réseau fixe à un réseau mobile. Plusieurs architectures ont été crées où la voix est combinée aux données et à l'imagerie pour être transportées sur un seul réseau *data.* La nature de ces réseaux ouverts a un impact sur la voix en terme de sécurité. D'où le besoin imminent de sécuriser la voix tout en assurant une bonne qualité de service à la voix aussi bien dans un réseau fixe que dans un réseau mobile ou IP. Des solutions de sécurisation sont proposées pour les données. Mais des solutions partielles voire incomplètes sont proposées pour la voix.

Dans un premier temps, nous définissons les besoins de la sécurisation de la téléphonie et les services de sécurité nécessaires à son déploiement. Ainsi, nous analysons la sécurité offerte par les différents réseaux offrant un service de téléphonie, à savoir la sécurité dans le réseau téléphonique traditionnel (RTC et RNIS), dans les réseaux mobiles (GSM et UMTS), et dans le réseau de données IP avec les deux architectures prépondérantes H.323 et SIP. Ceci va nous permettre de comparer les solutions de sécurité offertes par ces architectures de téléphonie et de pouvoir aborder d'une part, leur avantages et inconvénients et d'autre part, les exigences qu'ils ne peuvent pas satisfaire. Cette analyse nous conduit à un résultat éloquent qui est l'absence d'une solution de sécurité complète qui répond aux exigences de la téléphonie et qui permet d'effectuer un appel sécurisé de bout en bout.

Dans un second temps, nous proposons une architecture de sécurité pour une architecture de téléphonie abstraite. Cette architecture propose une nouvelle couche de service qui s'insère entre deux couches de niveaux N et N+1 du modèle de l'OSI. Ce choix permet d'assurer une transparence et une indépendance du réseau sous-jacent mais nécessite de revoir les interfaces et donc de définir une API qui s'occupe d'assurer cette transparence et l'interfaçage entre l'application de sécurité et le réseau sous-jacent. Cette architecture assure les services de sécurité et définit les politiques de sécurité nécessaires pour répondre aux exigences de la sécurisation des communications de voix.

Notre proposition de l'architecture de sécurité s'accompagne de la définition d'un protocole de sécurité que nous avons nommé SVSP pour *Simple Voice Security Protocol.* SVSP intègre les services de sécurité définis par cette architecture et permet la sécurisation d'un appel téléphonique de bout en bout. Des études ont été effectuées pour l'intégrer dans différentes infrastructures de téléphonie comme le réseau fixe, le réseau mobile GSM et une plate-forme de voix sur IP avec le standard H.323. Un prototype de SVSP a été implémenté suivi d'une intégration avec SIP, le standard de voix sur IP de l'IETF.

Abstract

Since the invention of the first telephone by Alexander Graham Bell in 1869, network telephony technology did not stop evolving: from circuit switching to packet switching, from fixed network to wireless network. Several new architectures were created which combines the transport of voice, data and image in the same data network. The nature of these open networks has an impact on the voice in terms of security. This yields to the imminent need to secure voice communications while insuring a good quality of service to the voice as well in fixed, wireless and IP networks. Different security solutions are proposed for the data. But partial even incomplete solutions are proposed for the voice.

First, we define the needs for securing the telephony and the security services required. Thus, we analyze the security offered by the different telephone networks, namely the security in the traditional telephone network (PSTN and ISDN), in the mobile networks (GSM and UMTS), and in the IP network based on the H.323 and SIP architectures. This will allow us to compare the security solutions offered by these telephony architectures and to be able to present their advantages and limitations and the security requirements that they cannot satisfy. This analysis drives us to an eloquent result that is the absence of a complete end to end security solution that complies with the security requirements of telephony.

Secondly, we propose security architecture for a unified telephony architecture. This security architecture proposes a service layer that is inserted between N and N + 1 layers of the OSI reference model. This choice provides a transparency and an independence of the underlying network but requires reviewing the interfaces and therefore the needs to define an API between the security application and the underlying network that insures transparency. This architecture provides the security services and defines necessary security policies to secure voice communications.

Following the security architecture, we defined a security protocol that we named SVSP for Simple Voice Security Protocol. SVSP satisfies the security services defined by this architecture that provides a secure end-to-end phone call. Studies were carried out to integrate it in different telephony infrastructures, namely with the traditional telephone network, GSM the mobile network and with the H.323 standard for voice over IP communications. A prototype of SVSP was implemented followed by integrating it with SIP the IETF voip standard.

Table des matières

Remerciements 4
Résumé 5
Abstract 6
Table des matières 7
Liste des Figures 11
Liste des Tables 13
1 Introduction....14
1.1 Pourquoi la sécurité de la téléphonie 14
1.2 Aperçu sur la sécurité de la téléphonie : 15
1.3 Motivations 17
1.4 Contribution de cette thèse 18
1.5 Organisation de ce mémoire de thèse 18
2 Etat de l'art et analyse de la sécurité de la téléphonie 20
2.1 Résumé 20
2.2 Introduction....20
2.2.1 Définition des vulnérabilités et des attaques de sécurité de la téléphonie .21
2.2.2 Quels services de sécurité pour la protection contre ses attaques 22
2.2.3 Problématique de la sécurisation de la téléphonie 24
2.2.4 Conclusion....24
2.3 Réseaux traditionnels analogique et numérique 24
2.3.1 Introduction....24
2.3.2 Architecture du réseau....25
2.3.3 Les vulnérabilités de la téléphonie classique et des PABX....26
2.3.4 La sécurité dans les réseaux de téléphonie traditionnelle 26
2.3.5 Conclusion....29
2.4 Réseau mobile GSM/GPRS 29
2.4.1 Introduction....29
2.4.2 Architecture du réseau....30
2.4.3 Vulnérabilités et failles du réseau GSM/GPRS 32
2.4.4 La sécurité dans le réseau GSM/GPRS 34
2.4.5 Conclusion....35
2.5 Réseaux UMTS 36
2.5.1 Introduction....36
2.5.2 Architecture du réseau....36
2.5.3 Vulnérabilités dans le réseau UMTS 37
2.5.4 La sécurité dans le réseau UMTS 37
2.5.5 Conclusion....38
2.6 ToIP / VoIP 38
2.6.1 Introduction....38
2.6.2 Architecture de la ToIP/VoIP 40
2.6.3 Vulnérabilités de la Téléphonie sur IP....40
2.6.4 La sécurité et la voix sur IP 42
2.6.5 : Une application de voip et sa sécurité 49
2.6.6 Conclusion....51

2.7 Analyse et comparaison des architectures de sécurité existantes 51
2.7.1 Solution de sécurité de bout en bout : FNBDT.....57
2.8 Conclusion générale du chapitre.....60
3 Architecture pour la sécurisation de la téléphonie.....62
3.1 Résumé 62
3.2 Motivations 62
3.3 Architecture abstraite du service de téléphonie 63
3.3.1 Architecture actuelle des réseaux de téléphonie.....63
3.3.2 Séparation Plan usager/Plan contrôle 63
3.3.3 Architecture abstraite de la téléphonie 64
3.4 Les services de sécurité pour la téléphonie.....65
3.4.1 Authentification de l'usager 65
3.4.2 Intégrité.....66
3.4.3 Confidentialité 66
3.4.4 Non rejeu 66
3.4.5 Non répudiation de l'appel.....66
3.5 Typologie des solutions, exigences et contraintes d'une sécurité de la téléphonie globale.....67
3.6 Les différentes architectures potentielles.....69
3.6.1 Architecture entre deux entités 69
3.6.2 Architecture entre 3 entités.....69
3.6.3 Architecture entre quatre entités.....70
3.6.4 Architecture active de la sécurité.....71
3.7 Placement des entités de sécurisation dans une architecture de téléphonie.....72
3.7.1 L'entité de confiance racine.....73
3.7.2 L'entité de confiance intermédiaire 73
3.7.3 Identifiant unique 74
3.7.4 Les cartes intelligentes.....74
3.8 Définition des politiques et leur impact sur les architectures 75
3.9 Les contrats de service.....77
3.10 Les définitions des services de bout en bout 78
3.11 API pour la sécurisation de la téléphonie 78
3.11.1 Le modèle d'appel SAPI.....79
3.11.2 Les fonctions de sécurité de cette architecture 83
3.12 Validation par un scénario abstrait de cette architecture 84
3.12.1 Analyse du modèle d'appel et de l'architecture 84
3.12.2 Scénario abstrait de l'appel.....85
3.13 Les caractéristiques de la conception de cette architecture 86
3.14 Conclusion: les avantages et inconvénients de cette solution.....87
4 Le protocole SVSP (Simple Voice Security Protocol) 88
4.1 Résumé 88
4.2 Motivations 88
4.3 Les étapes de conception du protocole SVSP.....90
4.4 Principales caractéristiques du protocole SVSP.....91
4.5 Le protocole SVSP 92
4.5.1 Architecture du protocole 92

4.5.2 Format du message SVSP 94
4.5.3 Les entêtes d'extension 95
4.5.4 Les messages d'extension d'entête 96
4.6 Les services de sécurités offerts par SVSP 98
4.6.1 Authentification 98
4.6.2 Confidentialité 100
4.6.3 Intégrité 100
4.6.4 Non-rejeu 100
4.6.5 Non-repudiation 101
4.7 Considération de gestion des clés 102
4.8 La sécurisation des communications entre les entités 102
4.8.1 Un scénario d'appel de base SVSP entre deux entités 103
4.8.2 Un scénario d'appel de base SVSP entre quatre entités 106
4.9 Les politiques de sécurité associées 109
4.10 Considérations de sécurité 110
4.11 Conclusion 111
5 Intégration de SVSP dans la téléphonie 112
5.1 Résumé 112
5.2 Intégration dans le réseau téléphonique commuté 112
5.2.1 Introduction 112
5.2.2 Présentation d'un appel type dans le réseau RTC 112
5.2.3 Intégration avec le réseau RTC 115
5.2.4 Intégration avec le réseau RNIS 116
5.2.5 Intégration avec le réseau SS7 117
5.2.6 Conclusion 119
5.3 Intégration dans le réseau GSM 119
5.3.1 Introduction 119
5.3.2 Présentation générale de l'appel 119
5.3.3 Architecture détaillée du réseau GSM 121
5.3.4 Limitations de cette intégration 122
5.3.5 Réalisation de cette intégration via un service de données 123
5.3.6 Conclusion 125
5.4 Intégration de SVSP avec H.323 125
5.4.1 Introduction 125
5.4.2 Présentation d'un appel entre deux postes H.323 125
5.4.3 Intégration avec H.323 129
5.4.4 Cas d'une conférence multi parties 131
5.4.5 Conclusion 132
5.5 Conclusion générale 133
6 Analyse et implémentation de SVSP 134
6.1 Résumé 134
6.2 Introduction 134
6.3 Analyse de SVSP 134
6.3.1 Les cartes intelligentes 134
6.3.2 Choix du vocodeur 136
6.3.3 Les limitations de SVSP 136

6.4 Mise en oeuvre de SVSP 139
6.4.1 Implémentation du prototype SVSP 139
6.4.2 Automate du protocole SVSP 148
6.5 Intégration de SVSP avec SIP 151
6.5.1 Les vulnérabilités de SIP 151
6.5.2 SVSP et SIP 152
6.6 Conclusion 153
7 Conclusion générale et perspectives 154
7.1 Conclusion générale 154
7.2 Perspectives et travaux futurs 156
Glossaire 157
Bibliographie 161
Annexe A : Publications associées à cette thèse 165
Annexe B : code source du prototype SVSP 166

Liste des Figures

Figure 1.1 - l'interconnexion de réseau de téléphonie 16
Figure 2.1 - Les protocoles de sécurité aux différents niveaux du modèle de l'OSI 21
Figure 2.2 - synoptique d'un réseau de télécommunication 25
Figure 2.3 – la ligne rouge entre le Kremlin et la Maison Blanche 27
Figure 2.4 – le produit STU-II 28
Figure 2.5 – Architecture du réseau GSM 31
Figure 2.6 – Architecture de sécurité dans GSM 34
Figure 2.7 - Architecture du réseau UMTS 36
Figure 2.8 - Périmètres comparés de la VoIP et de la ToIP. 39
Figure 2.9 - Convergence des réseaux voix-données. 40
Figure 2.10 - La pile H.323 avec les différents protocoles de sécurité. 42
Figure 2.11 – Pile protocolaire SIP 43
Figure 2.12 – EAP et 802.11 47
Figure 2.13 – le réseau Skype et ses différents éléments 50
Figure 2.14 – graphe simulant l'overhead de SRTP avec l'outil *Network Simulator (NS)* 56
Figure 2.15 - Les couches protocolaires de FNBDT. 58
Figure 3.1 - synoptique d'un réseau de téléphonie 65
Figure 3.2 - Téléphonie entre postes informatiques. 69
Figure 3.3 - Téléphonie entre poste informatique et téléphone. 70
Figure 3.4 - Téléphonie entre postes de téléphones. 71
Figure 3.5 - Architecture de sécurité 73
Figure 3.6 - synoptique des ressources de sécurité 76
Figure 3.7 - scénario de contrôle d'appel sécurisé de premier-parti 79
Figure 3.8 - scénario de contrôle d'appel sécurisé de tiers 79
Figure 3.9 - scénario de contrôle d'appel sécurisé de premier parti et de tiers 79
Figure 3.10 - modèle d'appel sécurisé à deux partis 80
Figure 3.11 - Machine à état de l'objet *SecureCall* 81
Figure 3.12 - machine à état fini pour l'objet *SecureConnection* 81
Figure 3.13 - machine à état fini pour l'objet *SecureTerminalConnection* 82
Figure 3.14 - scénario abstrait d'un appel téléphonique sécurisé 86
Figure 4.1 - L'architecture du protocole SVSP 93
Figure 4.2 - Les sous-couches du protocole SVSP 93
Figure 4.3 - Le format du message SVSP. 94
Figure 4.4 - format de l'entête d'extension 95
Figure 4.5 – machine à état de l'expiration des temporisateurs et de la retransmission 97
Figure 4.6 - Non-repudiation report 101
Figure 4.7 - Scénario d'un appel basic entre 2 entités 105
Figure 4.8 - Scénario d'un appel basic entre 4 entités 108
Figure 5.1 - Acheminement d'un appel en signalisation voie par voie 113
Figure 5.2 – la signalisation d'établissement d'appel entre deux commutateurs (NNI) 114
Figure 5.3 – intégration de SVSP dans un appel RTC 116
Figure 5.4 - Architecture de l'intégration de SVSP dans le réseau sémaphore 118
Figure 5.5 – modèle de référence du réseau SS7 119

Figure 5.6 - Représentation de la voix dans les différents plans de transmission.....121
Figure 5.7 - Signalisations d'appel H.323. 126
Figure 5.8 - conférence entre usagers H.323 contrôlée par un MCU 132
Figure 6.1 - Architecture d'une carte à puce 138
Figure 6.2 – scénario d'échange de message SVSP contenant un *Random* 141
Figure 6.3 – interaction entre les différentes classes Java 145
Figure 6.4 - certificat X.509 146
Figure 6.5 – rapport de non répudiation 146
Figure 6.6- Automate SVSP vu du client initiateur de la session.....149
Figure 6.7- Automate SVSP vu du client récepteur 150

Liste des Tables

Table 1- Ratio du *payload* sur la taille totale du paquet 55
Table 2 - Solutions et mécanismes de sécurité 61
Table 3 - Tableau des messages SVSP du scénario d'appel entre deux entités 106
Table 4 - Tableau des messages SVSP du scénario d'appel entre quatre entités 109
Table 5 – comparaison de *l'overhead* introduit par SVSP et autres protocoles de sécurité 137
Table 6 – valeurs du champ VoiceID du messages SVSP.....139
Table 7 – tableau des valeurs des champs de l'entête d'extension.....143
Table 8 – spécification technique et mesure des paramètres de performance 147
Table 9 – spécification technique 153

Chapitre 1

1 Introduction

1.1 Pourquoi la sécurité de la téléphonie

Jusqu'aux années 1990, le secteur des télécommunications était dominé par les opérateurs de réseaux téléphoniques fixes, traditionnels, nationaux qui généralement monopolisaient le marché. Les équipements provenaient principalement de quelques rares fournisseurs. Le lien qui relier les différents opérateurs et les équipementiers était très fermé, d'où des marchés fermés et donc sécurisés. L'avènement des technologies numériques à perturber considérablement ce schéma. La dérégulation [Wil99] devient une nécessité. Les nouvelles libertés acquises libèrent les exploitants de télécommunications des carcans dans lesquels ils opéraient. Ainsi, la déréglementation [Thi97] des années 1990 a provoqué un changement complet de la donne dans le secteur industriel et la restructuration des filières :

- les opérateurs de réseaux ont été contraints d'exercer leurs activités dans un cadre concurrentiel et non plus de monopole public et ce pour la partie la plus dynamique et, à terme, la plus importante de leur activité.
- les équipementiers très liés aux opérateurs ne bénéficiaient plus de marchés garantis et se sont retrouvés en concurrence avec ceux du monde de l'électronique.

Ces dernières années ont vu aussi l'émergence d'une nouvelle génération de systèmes mobiles qui ouvre la voix aux services multimédias et à la convergence entre la téléphonie (transport, accès, gestion de réseaux et de services et services support) et l'informatique (fourniture de services et de contenus). Cependant, les réseaux sans fils sont vulnérables aux écoutes et aux attaques par vol d'identité. La prise de mesures de sécurité se trouve nécessaire pour sécuriser le trafic de voix et de multimédia.

Ainsi, les réseaux de mobiles et les réseaux sans fil forment aujourd'hui un vaste environnement qui s'étend de la transmission jusqu'aux services et qui ont pour objectif d'offrir des services similaires au réseau fixe (voix) et voir plus (VPN, etc.). On se retrouve avec deux infrastructures (fixes et mobiles) qui ont tendance à se confondre, d'où la nécessite de converger les services (essentiellement la voix). L'évolution du réseau de téléphonie, aussi bien fixe que mobile, converge vers une infrastructure cohésive *data* qui combine la voix ou la téléphonie, les données, le fax et les transmissions vidéo, une infrastructure centrée sur le protocole IP qui s'étend sur le réseau de donnée – l'Internet. Les fournisseurs et les usagers mettent en avant la mutualisation de l'infrastructure réseau et celle des compétences des équipes téléphonie et réseau pour le déploiement de la ToIP et la VoIP au sein de leur entreprise et pour interconnecter des sites éloignés au site central et ceci au moindre coût. La ToIP prend d'assaut le grand public. Avec la banalisation des offres à des tarifs très agressifs

notamment dans le cadre des offres dites « *Triple Play* » chez les FAI (e.g, Free, Tiscali, etc.) et la montée en puissance d'offres comme celles de Phonesystem, Skype, Vonage ou Wengo, la ToIP a fait une percée remarquée dans le grand public [Bar05].

De nouvelles infrastructures de réseaux voient aussi le jour. La nouvelle génération de réseaux NGN accompagne les évolutions du monde des télécommunications en proposant une nouvelle architecture de services. Elle offre de grandes capacités de flexibilité, d'évolution, et d'ouverture aux réseaux et services d'opérateurs tiers. Cette évolution est fondée sur une dorsale paquets, avec le réseau IP, pivot permettant une évolution vers la convergence de réseaux, le multimédia et une interopérabilité avec les réseaux intelligents et les applications informatiques. Le succès du téléphone GSM a suscité des extrapolations dans le domaine de l'Internet grand public (besoins de services large bande pour mobile offert par l'UMTS).

Cette nouvelle convergence dans les réseaux, introduit de nouveaux avantages mais aussi des vulnérabilités au niveau des échanges, aussi bien de données que de la voix. La sécurité est devenu un aspect prioritaire pour les services d'une manière générale. Dans la mesure ou une grande partie de l'économie mondiale utilise ces infrastructures, se pose la question de sécuriser et de protéger cette infrastructure. C'est dans cet environnement que la convergence des réseaux et de la téléphonie fixe et mobile prend place. Ainsi, les *Phreakers* et les *Hackers* sont capables de compromettre et de contrôler différents aspects de la téléphonie, aussi bien les informations de signalisation que les paquets de voix. Des solutions de sécurisation sont proposées pour les données. Mais des solutions partielles voire incomplètes sont proposées pour la voix.

Le but de cette thèse est d'étudier l'état de l'art de la sécurité de la téléphonie, de pointer sur les avantages et les limitations des solutions existantes, et de sortir avec une proposition d'une architecture et d'un protocole de sécurité. Cette architecture permettra d'assurer un appel sécurisé de bout en bout tout en comblant les manques des solutions existantes.

1.2 Aperçu sur la sécurité de la téléphonie :

Dans les années 80, le téléphone sécurisé STU-III (*Secure Telephone Unit*) de la NSA [NSA] constituait une solution matérielle nouvelle de sécurité pour les communications de voix et de données. Il offrait un choix du niveau de sécurité des communications de voix et de données. De 1987 à 1999, la *General Dynamics* a équipé les bureaux et salles de conférence de la Maison Blanche, du Pentagone et des différents entrepreneurs de défense de par le monde, de plus de 240 000 postes téléphoniques STU-III [STU-III].

Au milieu des années 90, quatre niveaux distincts de sécurité avaient été définis pour les communautés d'utilisateurs spécifiques :

Type 1 : Le *Type 1* constitue le niveau le plus élevé de sécurité. Il a été exigé par le gouvernement américain pour la protection des informations classées Top Secret ;

Type 2 : La sécurité de *Type 2* a été prévue pour les informations sensibles mais non secrètes du gouvernement américain ;

Type 3 : La sécurité de *Type 3* a été conçue pour tous les autres besoins américains de sécurité domestique ;

Type 4 : La sécurité de *Type 4* a été définie pour la classe des produits sécurisés, à destination de clients internationaux.

Ces quatre groupes d'utilisateur pouvaient communiquer en toute sécurité au sein de leur communauté spécifique, mais ne pouvaient pas communiquer de façon sécurisée avec d'autres groupes d'utilisateurs. Par exemple, un utilisateur de *Type 1* ne pouvait pas avoir une conversation sécurisée avec un utilisateur de *Type 3* ou de *Type 4* parce que les produits sécurisés n'interopéraient pas.

Plusieurs facteurs majeurs réglementaires, économiques et géopolitiques ont récemment joué un rôle important, ce qui a poussé le gouvernement américain a éprouvé un besoin croissant d'intercommunication entre les différents niveaux de sécurité. D'autre part, comme les réseaux à fil, sans fil et ceux basés sur IP se développent d'une façon de plus en plus interopérable, et que chacun de ses réseaux apportent ses propres mécanismes de sécurité, la NSA, l'agence américaine en charge de la sécurisation des systèmes d'information, a lancé un certain nombre d'initiatives pour s'attaquer à jamais aux problèmes les plus durs de sécurité des réseaux, allant de l'interopérabilité d'Internet et la convergence des réseaux aux vulnérabilités des réseaux sans fil. Comme étant le plus grand environnement non protégé d'interconnexion des réseaux sur le globe, l'Internet est un défi formidable de sécurité et de nouvelles applications à valeurs ajoutées, dont l'une des plus en vogue est la voix sur IP. Il représente aujourd'hui pour beaucoup le seul plus grand marché de sécurité. Des efforts continuels au sein des groupes de travaux de l'IETF sont présents pour assurer la sécurisation de ce secteur, par des protocoles de sécurité de niveau réseau (i.e., IPsec [RFC2401]), ou des protocoles de sécurité de niveau transport (i.e., TLS [RFC2246], DTLS [DTLSdraft], SRTP [RFC3711], etc.) ou de niveau applicatif (i.e., S/MIME [RFC2633], SAML [SAMLdraft], etc.).

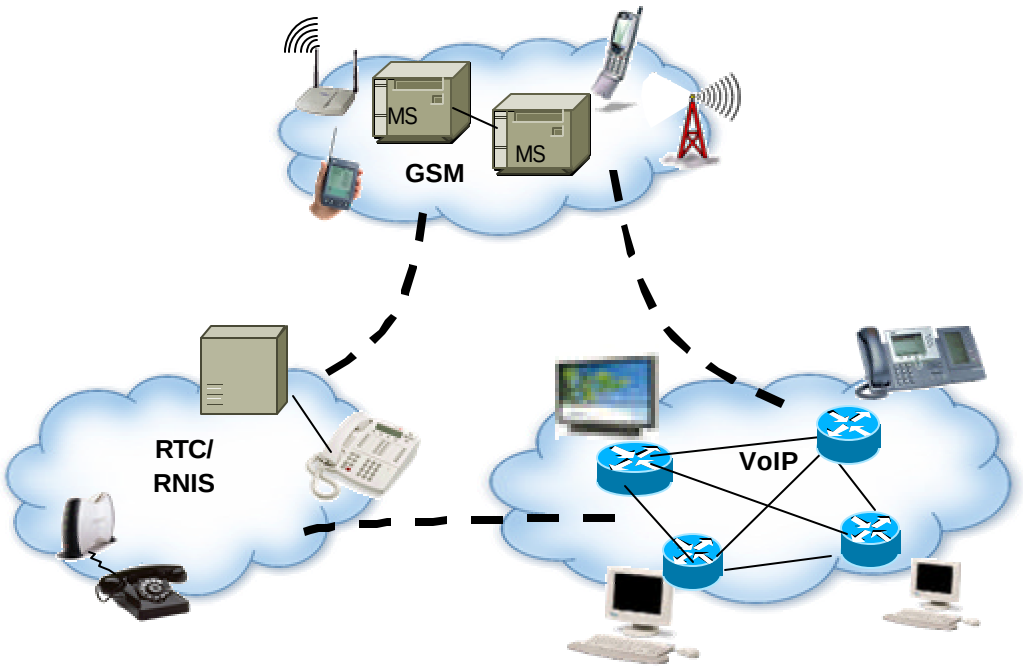

Figure 1.1 - l'interconnexion de réseau de téléphonie

Bien que les infrastructures de téléphonie citées précédemment apportent leurs propres solutions de sécurité, cependant ces solutions n'assurent pas une sécurisation de l'appel de bout en bout traversant différentes plates-formes de téléphonie.

En 1999, la NSA a réalisé une interopérabilité sécurisée entre les systèmes à fil et ceux sans fil lorsqu'elle a créé un consortium industriel et gouvernemental qui a convenu d'un protocole commun de signalisation, appelé FNBDT (*Future Narrow Band Digital Terminal*) [FNBDT]. Il a été également développé pour fournir un schéma de signalisation permettant aux utilisateurs de communiquer en toute sécurité avec d'autres produits compatibles.

1.3 Motivations

L'économie mondiale étant régie par le phénomène de la mondialisation, le monde des télécoms y cède et offre ses services. Les conséquences se reflètent au niveau technique et font ressentir le besoin urgent de trouver une solution de sécurisation de communications mondialement interopérables.

a. Motivations techniques

Vu la diversification et l'émergence continues de nouveaux réseaux, ajoutées à la *déréglementation*, les recherches se trouvent de facto orientées vers le développement d'une solution indépendante de l'infrastructure (fixe ou sans fil) et indépendante de la sécurisation offerte par les couches sous-jacentes. D'autant plus, que nombreuses sont les solutions qui permettent actuellement de se prémunir plus ou moins contre d'éventuelles attaques et, il y en a même des propriétaires, mais jusque là, d'aucune n'est polyvalente, notamment en matière de sécurité.

Actuellement, la sécurisation de la voix ne se fait pas de bout en bout. En effet, seuls les accès des IP Phone au réseau IP ou bien les échanges entre les terminaux mobiles et les réseaux sans fil sont sécurisés; tous les autres échanges ayant lieu en clair.

Par ailleurs, la sécurisation actuelle de la voix n'est pas uniforme. En effet, le protocole de sécurisation de la voix, utilisé dans le cadre d'une architecture H.323 [ITUT-H323-03] est différent de celui utilisé avec le protocole SIP [RFC2543] et tous deux diffèrent de celui en vigueur dans le cadre des réseaux radio mobiles.

b. Motivations économiques

Une des principales motivations économiques pour la sécurisation de la voix indépendamment de l'infrastructure, est la guerre de l'intelligence économique qui a lieu de façon bien discrète, voire secrète et dont les conséquences sont parfois redoutables !

À ce sujet, nous citons notamment l'existence du système Échelon[1], dont la révélation, il y a quelques années, a envenimé certaines relations au sein du monde diplomatique.

Ce système, mis en place pendant la Guerre froide, permet aux agences de renseignement des États-unis et de leurs alliés – la Grande-Bretagne, le Canada, l'Australie et la Nouvelle-Zélande – de passer au crible, trier, sélectionner et analyser à l'aide d'ordinateurs puissants, chaque jour, des millions de télécopies, de télex, de messages électroniques et d'appels téléphoniques du monde entier, à la fois par origine et

[1] MALLIÉ Cyril. Échelon : mode d'emploi. **In :** RFI. Échelon : mode d'emploi **[en ligne]**. Disponible sur : <http://cdcp.free.fr/dossiers/echelon/emploi.htm> (consulté le 22-10-2004).

selon un système de mots clés figurant dans des « dictionnaires » informatiques réactualisés chaque jour par les services secrets en fonction de leurs priorités du moment, mais également d'identifier une conversation où un sujet serait abordé à mots couverts.

Tous les réseaux de communication sont donc écoutés, des câbles sous-marins (des capteurs sont déposés par des plongeurs spécialisés) au réseau Internet (la surveillance du réseau mondial est particulièrement aisée : la quasi-totalité des données transitent par des « noeuds » situés sur le territoire américain, même lorsqu'il s'agit de connexions européennes !). D'où la nécessité urgente de trouver une solution qui sécuriserait de bout en bout et de façon fiable des communications entre réseaux hétérogènes et qui resteraient universellement interopérables.

1.4 Contribution de cette thèse

Motivé par la nécessité de *sécuriser les communications*, notre travail vient répondre aux nouveaux besoins de sécurisation de la téléphonie. Les principales contributions de cette thèse sont les suivantes :

a. Etat de l'art et analyse des solutions de sécurité existantes

Dans la première partie de cette thèse, nous faisons une étude critique des diverses solutions de sécurité dans plusieurs plates-formes de téléphonie. Nous analysons par la suite ces solutions et leurs différentes architectures. Nous définirons l'ensemble des critères de comparaison et des besoins en architecture de sécurité sur lesquels nous baserons notre proposition.

b. Proposition d'une architecture et d'un protocole de sécurité pour la téléphonie

Dans la deuxième partie, notre présentons notre contribution qui réside dans la conception d'une nouvelle architecture et d'un nouveau protocole de sécurité appelé SVSP (pour *Simple Voice Security Protocol*). Ce protocole a pour but d'offrir un cadre sécuritaire à la téléphonie indépendant du mode de transport sous-jacent, tout en prenant en compte la mobilité et un haut niveau de sécurité de l'usager. Il offre essentiellement le service d'authentification suivant deux modes : soit une authentification directe entre les usagers ou bien à travers une tierce partie de confiance et assure les services de sécurité nécessaire à la téléphonie.

c. Analyse de SVSP et intégration dans plusieurs architectures de téléphonie

Pour valider notre proposition, nous avons analysé les limitations et les possibilités d'intégration de SVSP dans plusieurs infrastructures de téléphonie les plus répandues comme le réseau fixe RTC, le réseau mobile GSM et la voix sur IP. Nous analysons SVSP en se basant sur les critères définis dans la première partie de cette thèse en le comparant aux solutions de sécurité existantes. Un prototype de SVSP est présenté avec une intégration avec SIP se basant sur la plate-forme JAIN-SIP [JAIN].

1.5 Organisation de ce mémoire de thèse

Cette thèse est organisée en trois parties principales : la première partie (chapitre 2) présente l'état de l'art de la sécurité de la téléphonie. Elle passe en revue les principales solutions de sécurité pour différentes infrastructures. Cette étude définit les

besoins et les caractéristiques des solutions de sécurité en particulier, pour les réseaux traditionnels, pour les réseaux sans fil et pour la voix sur IP. Elle nous permet de situer notre démarche dans le cadre des travaux menés jusqu'à présent et de dégager les limites et les faiblesses des solutions de sécurité existantes dans les différentes infrastructures de téléphonie.

La deuxième partie (chapitres 3 et 4) décrit notre analyse et notre proposition d'une architecture de sécurité pour la téléphonie dans le chapitre3. Nous définissons dans le chapitre 4 un nouveau protocole de sécurité qui est indépendant de toute infrastructure tout en garantissant la mobilité de l'usager en utilisant des cartes à puce. Cette approche est suffisamment flexible pour prendre en compte les différents types d'authentification, de contrôle d'accès par des politiques de sécurité et de protection d'identité. Elle offre deux types d'authentification, une reposant sur les certificats entre les usagers et une autre sur une clé pré-partagée entre l'usager et une tierce partie de confiance.

La troisième partie (chapitres 5, 6) présente une étude approfondie pour intégrer et évaluer SVSP. Dans le chapitre 5, nous étudions les possibilités et les limitations d'intégration de SVSP dans plusieurs infrastructures de téléphonie. Pour cela, nous avons choisi d'étudier cette intégration avec les plates-formes de téléphonie les plus répandues sur le globe terrestre telles le réseau téléphonique commuté RTC, le réseau mobile GSM et la voix sur IP avec le standard H.323. Le chapitre 6 s'intéresse à analyser SVSP, son impact sur la téléphonie et présenter une implémentation d'un prototype de SVSP. Enfin, dans le chapitre 7, nous concluons sur l'intérêt de notre travail et ses perspectives.

Le lecteur trouvera dans l'annexe A la liste des publications associées à cette thèse, et dans l'annexe B le code source de l'implémentation du prototype de SVSP.

Chapitre 2

2 Etat de l'art et analyse de la sécurité de la téléphonie

2.1 Résumé

La parole est l'entité principale pour permettre aux personnes de communiquer ensemble. Elle est transportée par différents moyens comme l'air. Pour permettre à des personnes distantes de communiquer, les réseaux de télécommunications ont été construits. Les technologies utilisées ont beaucoup évoluées dans ces dernières décennies. Cependant, ses moyens de transport n'assurent pas la confidentialité des conversations entre les communicants. Pour assurer la sécurité des ces communications, différents mécanismes sont nécessaires pour sécuriser cet échange. Le but de ce chapitre est de présenter un état de l'art de la sécurité offerte par les différents réseaux de communications pouvant transporter la parole et d'analyser cette sécurité.

2.2 Introduction

L'évolution des télécommunications répond principalement à une double attente : la convergence des réseaux (fixe/mobile et voix/données) et l'intégration de services multimédias. Le déploiement et l'exploitation de services de télécommunications dépendent de l'infrastructure et des supports offerts par les réseaux sur lesquels ils sont implantés. Ainsi, la qualité de services (QoS pour *Quality of Service*) identifiée par un ensemble de paramètres (délais, débits, taux de perte…) et disponible sur le réseau est une des conditions préalables à la fourniture du service. Les évolutions des télécommunications vers la téléphonie mobile et spécifiquement la voix sur IP supportée par les avancées technologiques ont fait prendre conscience que fournir la téléphonie avec une certaine QoS n'est plus suffisante. La sécurité de ces réseaux et des applications qui y tournent constitue actuellement une nouvelle condition et un souci prépondérant pour les opérateurs de télécommunications mais aussi pour les fournisseurs de service et pour les particuliers. Les enjeux et priorités sont donc liés à la fourniture de services de téléphonie et de multimédias en conjonction avec l'exigence de mobilité, d'accessibilité et de sécurité.

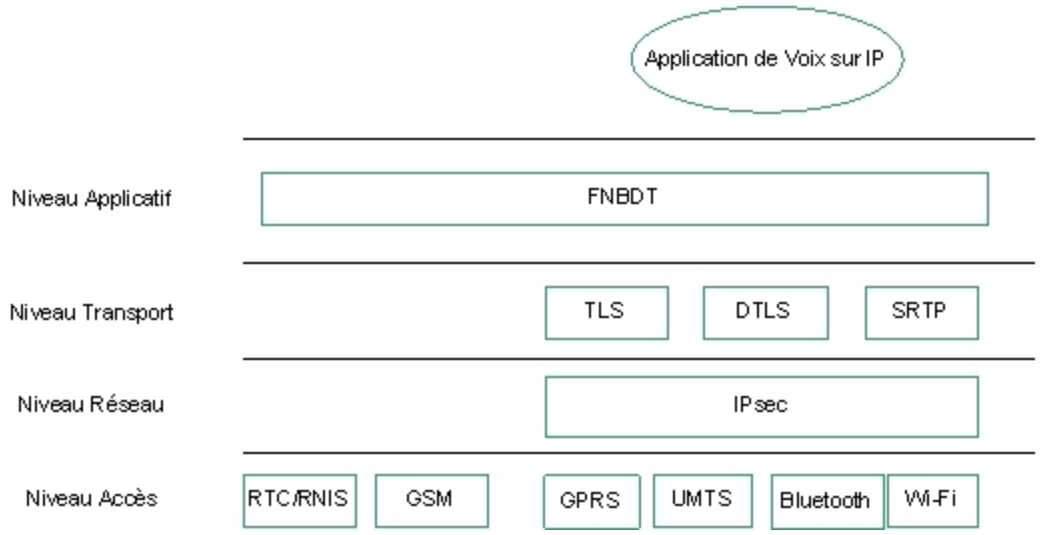

Figure 2.1 - Les protocoles de sécurité aux différents niveaux du modèle de l'OSI

Dans ce chapitre, nous proposons de présenter l'état de l'art de la sécurité des réseaux téléphoniques dans l'ensemble des contextes fixe, mobile et data. Pour cela nous débuterons par définir d'une manière globale les vulnérabilités des réseaux de téléphonie ainsi que les services de sécurité essentiels qui seront suivis par un état de l'art de la sécurité de la téléphonie dans différentes infrastructures de téléphonie. Le foisonnement des réseaux d'accès fixe (RTC/RNIS), de technologies d'accès radio (GSM, GPRS, UMTS) ou de technologies d'accès sans fil (Bluetooth, WLAN) offre chacun des mécanismes de chiffrement différent comme nous le verrons par la suite. Cette évolution des réseaux de télécommunications porte aussi vers une évolution des réseaux de transport actuels vers un réseau de transport IP, multiservice et haut débit. Ce réseau de transport IP apporte avec lui ses propres protocoles et mécanismes de sécurité et ceci à différents niveaux du modèle de l'OSI. Ainsi, nous trouverons le protocole IPsec de niveau réseau qui accompagne le protocole IP et le sécurise. Si nous passons au niveau transport, nous verrons différents protocoles de sécurité qui assurent la sécurisation aussi bien du trafic UDP que le trafic TCP avec les protocoles respectifs DTLS et TLS. Si nous passons vers les couches applicatives, nous trouverons le protocole RTP qui achemine les échantillons de la voix sur un réseau tout IP et qui est accompagné par son profile de sécurité, le protocole SRTP. Une solution de sécurité, le protocole SCIP (ou FNBDT), de niveau applicatif est proposée par la NSA pour sécuriser le trafic de voix et de données indépendamment du réseau sous-jacent.

2.2.1 Définition des vulnérabilités et des attaques de sécurité de la téléphonie

Avant d'aborder les fonctions de sécurité implantées dans les réseaux téléphoniques actuels, il est nécessaire de mentionner les menaces qui affectent la télécommunication de nos jours. N'importe quel système de télécommunication, et pour que sa sécurité soit acceptable, doit pouvoir se défendre contre tout genre de menaces.

En effet, nous pouvons rencontrer au moins cinq failles dans le réseau téléphonique définies comme suit :

A- écoute des conversations : Un attaquant ou espion passif écoute les données sans essayer de les changer. Il est très dangereux car on ne peut pas détecter son écoute que s'il utilise ultérieurement les informations captées. L'exemple le plus important de cette menace est l'écoute des conversations entre les hommes politiques et entre les pays pour obtenir des informations secrètes. Lorsque ces informations seront publiées, la victime saurait que ses conversations ont été écoutées.

B- Tracer les déplacements d'un usager : La sécurité d'un système de télécommunication mobile n'est pas seulement reliée à la protection des données de l'usager (voix, …). Elle doit garantir en plus une protection des informations de localisation, car un espion peut utiliser ces informations pour tracer les mouvements de l'utilisateur sur une carte.

C- Modifier les données : Un espion qui a pu avoir accès au canal de données, peut non seulement écouter les informations mais aussi modifier les données transmises. Par exemple, avoir accès au logiciel de configuration et de maintenance du réseau téléphonique ou accéder à la messagerie vocale ou l'utilisation d'un enregistreur relié au lien physique ou bien effectuer une demande de transfert d'argent peut être transformée vers le compte de l'attaquant.

D- Rejeu : Pour certains types d'applications, essentiellement la VoIP, l'espion peut enregistrer l'information et la retransmettre pour son bénéfice. Par exemple, une combinaison d'un nom et d'un mot de passe peut être captée puis transmise vers le compte de l'attaquant, ou bien le rejeu d'une conversation.

E- Accéder aux services avec l'identité d'autrui : L'attaquant peut jouer le rôle d'une autorité et persuader un employé d'une entreprise de lui donner son mot de passe pour s'autoriser l'accès à la configuration et à la maintenance du PABX de l'entreprise. Une bonne authentification peut défendre contre cette menace.

2.2.2 Quels services de sécurité pour la protection contre ses attaques

Le transfert d'information critique sur toute infrastructure de communication accessible au publique présente des vulnérabilités au niveau de l'authenticité et de la confidentialité des informations acheminées sur ces réseaux. On a notamment ressentit l'impact de ces vulnérabilités avec l'arrivée du réseau Internet puisque c'est le réseau le plus ouvert.

Les requis de la sécurité de la voix nécessite le déploiement de mécanismes de sécurité tel l'authentification, la confidentialité, l'intégrité, la non-répudiation, et le non-rejeu. Ces mécanismes font appel à un certain nombre de concepts. Chaque concept possède ses propres algorithmes et protocoles. Au-delà de ces services existe un grand nombre de protocoles et d'algorithmes qui s'appliquent ou non à la voix selon les options retenues et leur impact de point de vue performance (*overhead* protocolaire, temps de traitement, etc.) sur la qualité de la voix.

Commençons par définir succinctement ces mécanismes :

- *Authentification* : elle consiste à garantir l'identité de la personne qui envoie le message. Actuellement elle est assurée par l'échange de certificats. Les mécanismes d'authentification sont similaires à ceux de l'intégrité. Sauf que le principe se base sur des fonctions de hachage avec clé secrète. L'empreinte (digest) obtenue est appelée *Message Authentication Code* (MAC). On aura alors les fonctions de hachage avec clé suivantes : HMAC-MD5 et HMAC-SHA1 [RFC2104]. Les certificats se veulent l'équivalent des empreintes digitales dans le monde physique. Ils doivent permettre d'affirmer l'identité d'une personne de façon infaillible. Au certificat numérique est associé une autorité de certification (CA) qui le délivre. Cette autorité de certification est un tiers de confiance qui est considéré comme fiable par la totalité des utilisateurs.
- *Confidentialité* : elle représente le fait qu'une conversation ne puisse être comprise que des personnes concernées. Cette fonction est assurée par le chiffrement des données. Deux approches principales existent pour assurer la confidentialité : architecture à clé symétrique et celle à clé publique. Le principe du chiffrement à clé symétrique repose sur l'utilisation d'une même clé pour les opérations de chiffrement et déchiffrement. Les algorithmes à clé symétrique les plus utilisés sont RC-4 [RC4], DES [DES], Triple DES [3DES] et AES [AES]. Le second principe est le chiffrement par clé publique connu aussi par chiffrement asymétrique où deux clés différentes sont utilisées : l'une est publique (non secrète) distribuée à toutes les personnes qui souhaitent la connaître et est utilisée pour le chiffrement des messages, l'autre clé est privée connue uniquement par le récepteur. Les deux clés sont liées par une relation mathématique à sens unique qui fait que toute personne qui connaît la clé publique d'une autre personne ne peut en déduire sa clé privée. Cette méthode est utilisée par les algorithmes de création des signatures numériques. Il existe plusieurs standards pour la génération des clés de chiffrement. Nous citons à titres d'exemple RSA [RSA] et Diffie-Hellman [Dif76] qui sont les plus répandus.
- *Intégrité* : la vérification de l'intégrité d'un message consiste à s'assurer que les données n'ont pas été modifiées entre l'envoi du message et sa réception accomplie par une fonction de hachage unidirectionnelle. Cette fonction va produire à partir d'un message de longueur variable un message de longueur fixe (empreinte), généralement de taille plus petite et qui est caractéristique du message initial. On ne peut trouver une même empreinte appartenant à deux messages différents. Parmi les fonctions de hachage, nous citons MD5 [RFC3121] et SHA-1 [SHA93].
- *Non–répudiation* : c'est le moyen de garantir qu'un échange de données a bien eu lieu de sorte que la personne ne peut se rétracter par la suite.
- *Non-rejeu* : consiste à mémoriser les données par un adversaire puis de les réinjecter dans le réseau. La protection contre le rejeu est possible si le mécanisme d'intégrité est présent. Il est souvent offert avec l'authentification.
- La *disponibilité des ressources* peut être considérée comme un mécanisme de sécurité afin de minimiser les délais et les rejets de services. Cette fonctionnalité sera du pouvoir des fournisseurs qui devraient assurer un certain degré de disponibilité des ressources déployées.

2.2.3 Problématique de la sécurisation de la téléphonie

Suite à la convergence des réseaux de données et de téléphonie et à l'interconnexion de ces derniers à l'Internet où la *voip* prend place, la problématique de la sécurisation de la téléphonie se pose alors puisqu'il devient plus évident aux téléphonistes que les attaquants puissent aussi exploités les failles de leurs réseaux et alors, les attaques pourront tout aussi bien prendre naissance dans les réseaux de téléphonie non IP. Les mesures de sécurité prises dans ces réseaux de téléphonie ne sont pas draconiennes; alors que les solutions de sécurité existantes sont génériques et donc répondent à moindre mesure aux spécificités et aux contraintes de la voix sur les réseaux de donnée. La qualité de la voix - qui est affectée par le délai, la variation de délai, les pertes de paquets - devrait être respectée. Des initiatives existent notamment à l'IETF pour proposer des filtres dynamiques [RFC3303] pour la protection contre la prolifération de virus et la limitation des attaques de sécurité. Ces filtres prennent en compte la dynamique des allocations des ports UDP et TCP pour la téléphonie sur IP mais leur objectif n'est pas d'offrir des services de sécurité.

D'autre part, il est bien connu que si nous introduisons de la sécurité, nous sommes alors entrain d'augmenter le nombre d'opérations et *l'overhead* protocolaire. En effet, le nombre d'opérations peut se diviser entre le nombre de messages pour échanger les paramètres de sécurité et les opérations cryptographiques. Ces opérations sont gourmandes en ressources, essentiellement en mémoire et augmentent la charge du CPU. *L'overhead* protocolaire est dû aux champs supplémentaires ajoutés au message (ou paquet) initial et qui sont nécessaires aux services de sécurité. Ainsi, nous pouvons alors estimer l'impact de la sécurité sur la QoS de la voix, spécifiquement en termes de délais et de charge cryptographique. Nous analyserons l'impact de la sécurité apportée par la contribution de cette thèse sans rentrer dans les détails de son impact sur la QoS de la téléphonie puisque la QoS ne fait pas partie de ce mémoire.

2.2.4 Conclusion

Dans cette section du chapitre, nous avons essayé d'introduire les vulnérabilités du réseau téléphonique ainsi que certaines types d'attaques lancées sur le réseau de téléphonie fixe et mobile. Nous avons aussi décrit les services de sécurité qui permettent de défendre le réseau contre ces attaques. Dans la suite du chapitre, nous allons voir un ensemble de réseaux de téléphonie, les attaques et les solutions de sécurité implantées.

2.3 Réseaux traditionnels analogique et numérique

2.3.1 Introduction

Le service téléphonique a été le premier service de télécommunication à avoir été déployé dans les réseaux publics. La conversation téléphonique comporte des signaux de signalisation pour l'établissement de l'appel et les signaux vocaux. La téléphonie est ainsi séparée en deux plans, un plan usager par lequel circulent les échantillons de voix en provenance de l'écouteur du poste téléphonique et le plan contrôle où s'effectuent l'échange de la signalisation pour l'établissement et la terminaison de l'appel. L'accès au plan usager vulnérabilise les conversations téléphoniques alors que l'accès aux informations de signalisation rende certaines informations sur l'usager disponibles aux

attaquants comme l'identité de l'usager et autres informations critiques. Nous verrons par la suite quelles sont les vulnérabilités de la téléphonie classique et quels sont les moyens mis en place pour assurer la sécurité des conversations.

2.3.2 Architecture du réseau

Relier chacun des abonnés à tous les autres nécessiterait un écheveau de milliards de liaisons, ce qui est irréalisable. Aussi des aiguillages entre les différents niveaux de jonction du réseau sont réalisés par des commutateurs qui relient les terminaux (téléphones) entre eux. La commutation [Rig98] n'est pas uniquement la connexion d'une ligne appelante à un moyen banalisé de transmission, c'est aussi la réception et l'émission de signalisations et l'exécution de la logique du service de traitement des appels.

La signalisation est constitué des messages de gestion des appels, et permet la supervision des procédures telles que le traitement d'appel ou la fourniture de services avancés (rappel sur occupation, numéro universel, etc.) La signalisation sémaphore (ou SS7 S*ystème numéro 7*) [Rig98] définit la transmission de la signalisation sur les réseaux téléphoniques et décrit les couches et protocoles utilisés dans l'architecture du réseau. Il est indépendant du réseau d'acheminement du trafic (ou du plan usager). Le réseau sémaphore est donc équipé de commutateurs de paquets spécialisés (STP) à haut débit et à faible temps de traversée, qui ne font rien d'autre que déterminer le nœud suivant vers lequel les messages sémaphores doivent être routés et s'assurer de leur livraison dans le bon ordre.

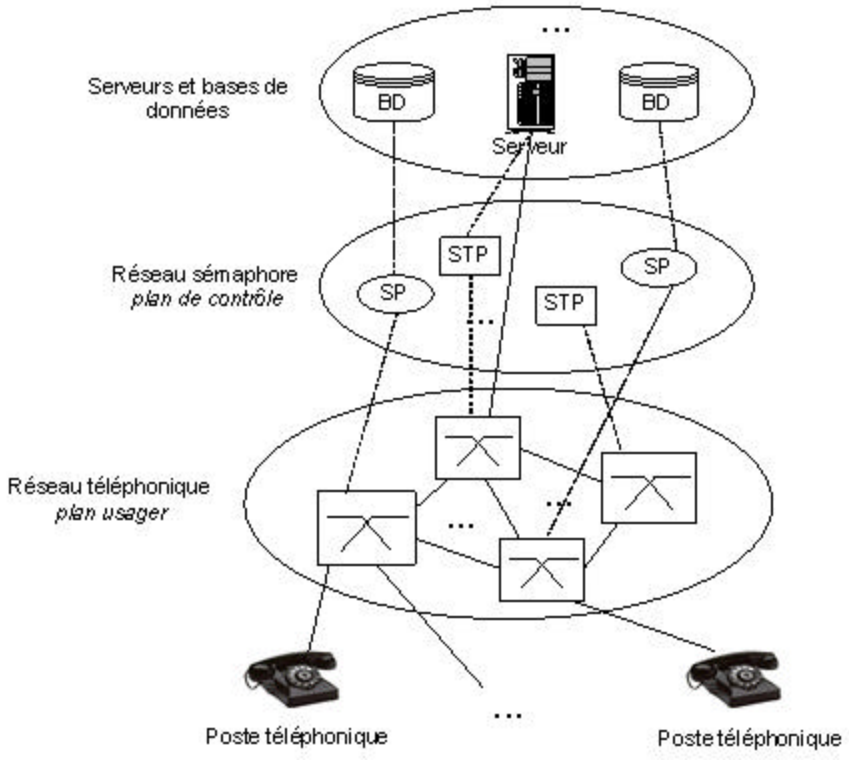

Figure 2.2 - synoptique d'un réseau de télécommunication

Ce réseau peut également inclure des équipements de fourniture et de gestion de services tels que les serveurs de messagerie (vocale et texte) et un réseau intelligent (IN pour *intelligent network*)

2.3.3 Les vulnérabilités de la téléphonie classique et des PABX

La sécurité des communications dans le réseau téléphonique classique était longtemps tenue pour acquise alors que la réalité était loin de l'être. En effet, il suffisait d'une capacité et l'accès physique à un fil pour « casser » cette sécurité. Sans parler de la surveillance des conversations par le gouvernement pour des raisons de sécurité étatique nationale et internationale. Une attaque célèbre lancée contre le réseau téléphonique est connue sous le nom de «*Phone Phreaking* ». Elle consistait à reproduire une tonalité à 2600 Hz capable de lancer des opérations comme appeler un usager juste pour écouter les messages d'erreur générés par le commutateur public. Un des pilons du « *phreaking* » est M. Joe Engressia [Phreak] qui a découvert et développé cette attaque.

En effet, nous pouvons rencontrer au moins quatre failles dans le réseau téléphonique traditionnel présentées comme suit :

- Ecoute illicite par accès physique aux fils (*wire tapping*) ;
- Vol de services (*phone phreaking*) ;
- accès illicite, local ou à distance, au logiciel d'exploitation contrôlant le commutateur, au logiciel de configuration, d'administration et de maintenance du réseau téléphonique ;
- accès à la messagerie vocale ou l'utilisation d'un enregistreur relié au lien physique ;
- utilisation d'un émetteur de fréquence radio avec microphone, connecté au lien physique ;
- accès illicite aux bases de données (création et modification des bases de données administratives, des utilisateurs, de facturation, etc.) ;
- modification de la mise à jour du système d'exploitation des commutateurs (injection de programme corrompu)
- exploitation malveillante des caractéristiques de l'utilisateur (renvoi d'appel, etc.)

2.3.4 La sécurité dans les réseaux de téléphonie traditionnelle

Sécuriser le réseau téléphonique revient à sécuriser aussi bien la conversation entre usagers mais aussi les messages de signalisation qui sont nécessaires pour l'établissement de l'appel et qui transportent des informations critiques sur l'usager.

La sécurisation de la téléphonie traditionnelle requiert outre l'utilisation d'équipements de cryptage pour chiffrer les conversations entre les usagers une sécurisation de l'architecture du plan contrôle et donc du réseau de signalisation. Nous verrons par la suite la sécurité telle offerte par le réseau analogique (RTC pour *Réseau Téléphonique Commuté*) et par le réseau numérique (RNIS pour *Réseau Numérique à Intégration de Service*)

2.3.4.1 Réseau Téléphonique Commuté (RTC)

L'authentification dans le réseau RTC est une authentification non cryptographique assurée par l'envoi du numéro de la ligne (Caller ID) qui identifie uniquement la ligne mais pas l'utilisateur de la ligne. Les services de sécurité que l'on trouve dans le réseau téléphonique commuté sont essentiellement la confidentialité des conversations téléphoniques. Ainsi pour contrer les failles citées dans le paragraphe précédent, la meilleure solution est d'utiliser des produits qui permettent de chiffrer les communications téléphoniques, les faxes, la transmission de fichier électronique. Le meilleur exemple que l'on peut citer est la ligne rouge établie entre le Kremlin à Moscou et la Maison Blanche à Washington.

Figure 2.3 – la ligne rouge entre le Kremlin et la Maison Blanche

Les mécanismes de sécurité utilisés pour sécuriser la transmission de la voix se divisent en trois catégories :

- brouillage simple par manipulation temporelle ou fréquentielle du signal analogique. Une autre technique repose sur la subdivision du signal en plusieurs niveaux puis la retransmission du signal dans différents niveaux fréquentiels.
- brouillage par code glissant et cryptage analogique se basant sur des techniques de brouillage et de cryptage plus complexe.
- cryptage en utilisant des algorithmes pour coder les signaux numériques par transposition de bit et/ou par opérations de substitution. Cette technique s'applique à un signal vocal analogique après sa numérisation.

Si nous voulons faire une rétrospective rapide sur l'évolution des équipements de chiffrement des communications entre les usagers, nous commencerons par :

- le system de voix sécurisé *SIGSALY* [Sigsaly] qui a été utilisé durant la seconde guerre mondiale. Ce système inclut des vocodeurs, une transmission sans fil, une distribution de clé, une synchronisation pour le cryptage, etc. ;
- le système *STU-I* qui utilisait un ordinateur avec un processeur de signaux numériques ;
- le système *STU-II* qui assure la sécurisation de la voix à un débit de 2400 à 9600 bps ;

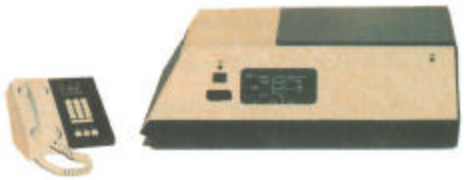

Figure 2.4 – le produit STU-II

La photographie ci-dessus (fig. 2.4) montre un produit *STU-II*. Plutard, il formera la base du standard NATO *NBSV-II* [NSA] qui permettra une interopérabilité entre les produits de sécurité.

Le produit *STU-III* a été développé par la *National Security Agency* (NSA) en 1987. Il utilise des clés symétriques pour le chiffrement des messages vocaux. Ces clés sont installés et sauvegardées dans les appareils de téléphones. Il possède aussi un système de gestion de clé propre à lui. Actuellement, la NSA est sorti avec le protocole FNBDT pour sécuriser les communications de données et de voix sur des liens à très faibles débits que nous verrons ultérieurement dans ce chapitre.

L'intégrité et le non rejeu sont deux services de sécurité qui ne sont pas pris en considération dans les solutions de sécurité offertes pour ce réseau. Nous dirons que le service de non-répudiation non cryptographique de l'appel est disponible en recourant aux factures détaillées issues par l'opérateur. Mais l'usager doit se fier à la parole de l'opérateur.

2.3.4.2 Réseau numérique (RNIS)

Le réseau numérique RNIS a été défini comme une solution globale de communication assurant les services multimédia (voix, données et vidéo). Comme le réseau traditionnel de la voix, ce réseau peut être la voie à des actes d'écoute sur les canaux de communications, d'accès aux informations critiques traversant jusqu'au réseau local. Des efforts ont été notés pour la définition des besoins de la sécurité dans ce réseau [ISDN]. Les services de sécurité sont similaires à ceux offerts dans le réseau RTC, sauf que le matériel pour le chiffrement du RTC n'est pas compatible avec le réseau RNIS du fait de l'architecture de ce réseau numérique. Des recherches ont été effectuées et un prototype de sécurité de niveau physique a été développé pour assurer la confidentialité d'utilisateur à utilisateur en appliquant le standard *TripleDES* comme algorithme de cryptage et une authentification de l'utilisateur. Ce prototype a été intégré dans l'unité de terminaison (NT1) entre le point de référence S et le point de référence U.

A l'instar des produits STU-II et STU-III du réseau traditionnel, d'autres efforts ont été effectués pour sortir avec des produits RNIS sécurisés offerts par 4 constructeurs tels le *TSK-3 Secure ISDN Phone*, *STE*, *BRENT* et *TopSec-703*. Cependant, ses efforts n'ont pas abouti à une interopérabilité entre les différents produits sur le marché comme elle est offerte par le standard *NBSV-II*. Ainsi sans un standard de sécurité, nous obtiendrons alors :

- De multiples protocoles de communications
- De multiples protocoles de sécurité
- De multiples suites cryptographiques
- De multiples systèmes de gestion de clé

Pour répondre à tous ces points, de nouvelles approches de sécurité devraient être considérées pour partager une architecture de sécurité commune.

2.3.4.3 Sécurisation à travers le réseau de signalisation SS7

Afin de sécuriser les communications de voix, une proposition d'une architecture de sécurité [Shar02] a été faite à l'Université de Georges Mason pour assurer la confidentialité des conversations et l'authentification des usagers. Cette proposition porte sur une sécurité de niveau applicatif et de bout en bout avec une modification minimale de l'infrastructure du réseau téléphonique public et une intégration du service de distribution des clés public aux téléphones. Ce modèle se base sur le chiffrement symétrique des signaux de voix entre les téléphones avec une clé symétrique à une seule utilisation unique. Ceci permet de prévenir les attaques par rejeu. Cette étude propose aussi des mécanismes de control d'accès des usagers et des téléphones utilisés pour assurer les communications sécurisées. La technique d'authentification proposée est basée sur l'utilisation de cryptographie par clé publique. Elle offre au centre d'authentification la garantie de l'authenticité du téléphone à l'autre bout de la connexion.

Cette proposition assure l'authentification des usagers vis-à-vis du réseau et vice versa. Son avantage est qu'elle utilise le réseau de signalisation SS7 comme support de déploiement pour pouvoir atteindre le plus large nombre de réseaux de téléphonie et donc d'utilisateurs. Son implémentation nécessiterait la modification du réseau actuel pour supporter les différentes entités mises en jeu.

2.3.5 Conclusion

Nous voyons bien que dans le réseau téléphonique traditionnel, la majorité des services de sécurité sont quasi-inexistantes. Seule la confidentialité prenait la part du marché de sécurité pour assurer le chiffrement des conversations entre les usagers. Les produits n'assuraient pas une interopérabilité des communications. D'où la nécessité d'un protocole de sécurité standardisé. Récemment, une proposition de sécurité a été faite qui repose sur l'utilisation de certificats à clés publiques déployées dans le réseau de signalisation. Cependant, assurer un niveau de sécurité introduirait une augmentation des délais de transmission due aux échanges pour assurer l'authentification des usagers et au cryptage des données de voix. Un compromis entre niveau de sécurité et qualité de la communication s'impose.

2.4 Réseau mobile GSM/GPRS

2.4.1 Introduction

Le réseau mobile GSM [Lag00], destiné à transmettre de la voix en mode circuit, a bénéficié du système GPRS, qui utilise une technologie d'accès radio proche de celle de GSM et un système de transport de données en mode paquet avec le vecteur IP au niveau du cœur du réseau, pour faire évoluer son offre à des services de transmission de données en mode paquet et donc pouvoir offrir un service de voix sur IP.

Dans un réseau de téléphonie fixe, à un numéro correspond une adresse physique fixe (une prise de téléphone), alors pour le réseau GSM/GPRS, le numéro d'un terminal mobile est une adresse logique constante à laquelle il faut associer une adresse physique

qui varie suivant les déplacements de l'usager du terminal. La gestion de cette itinérance nécessite la mise en œuvre d'une identification spécifique de l'utilisateur. De plus, l'emploi d'un canal radio rend les communications vulnérables aux écoutes et aux utilisations frauduleuses. La fraude des réseaux cellulaires, incluant le clonage et le piratage (*hijacking*) des appels de voix coûtera des billions de dollars par an. Les attaquants (*hackers*) ne volent pas uniquement des services, et violent la confidentialité des appels, mais aussi ils étaient capables de pénétrer dans les systèmes informatiques des opérateurs et de compromettre les bases de données de facturation et d'informations des appels.

2.4.2 Architecture du réseau

Pour gérer les spécificités des communications avec les mobiles, le GSM a introduit des équipements spécifiques n'existant pas dans réseaux téléphoniques fixes classiques RTC (ou *PSTN*). Ces équipements, et plus généralement toutes les fonctions relatives à la gestion des utilisateurs mobiles, ont été regroupés dans un type de réseau particulier, appelé PLMN. Ces réseaux se raccordent aux PSTN et aux PLMN d'autres opérateurs par des passerelles spécifiées dans le standard GSM.

Nous ne rentrerons pas dans les détails, nous nous limiterons à donner une vue d'ensemble de l'architecture du réseau GSM/GPRS en mettant en relief certaines parties en relation avec la sécurité dans GSM/GPRS.

a. **Le sous-système radio ou BSS** regroupe deux équipements impliqués directement dans la sécurité dans GSM, qui sont la MS et la BTS.
   - MS (*Mobile Station*) ou terminaux mobiles : Chaque terminal est équipé d'une carte à puce, ou carte SIM (*Subscriber Identity Module*). Cette carte contient les informations relatives à l'abonnement de l'utilisateur, telles que ses identités, l'IMSI (*International Mobile Subscriber Identity*) et le TMSI (*Temporary Mobile Subscriber Identity*), et ses algorithmes de chiffrement. Le terminal possède sa propre identité, l'IMEI (*International Mobile Equipment Identity*), qui permet de connaître, par exemple, l'identité du constructeur du terminal mais aussi d'empêcher l'utilisation de terminaux volés ou non conformes.

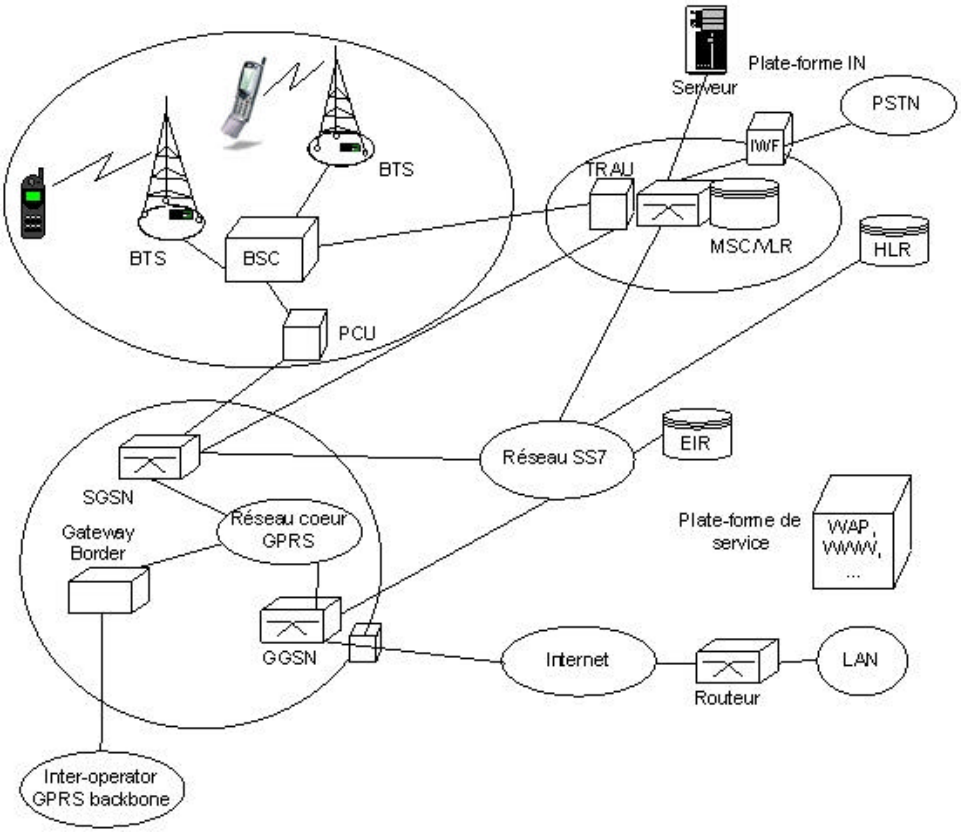

Figure 2.5 – Architecture du réseau GSM

- BTS (*Base Transceiver Station*) ou stations de base : les BTS forment les points d'accès au réseau GSM des utilisateurs mobiles. Elle authentifie et chiffre le trafic entre elle et le terminal mobile.
- BSC (*Base Station Controller*) ou contrôleurs de BTS. : les BSC assurent le contrôle d'admission des appels, la gestion des *handovers* ou le contrôle de puissance est organisé par le BSC.

b. **Le sous-système réseau ou NSS** s'occupe de l'interconnexion avec les réseaux fixes, publics ou privés, auxquels est rattaché le réseau mobile. Il gère aussi l'établissement des communications avec les utilisateurs mobiles, utilisateurs dont il détient un profil. Ses éléments sont :

- MSC (*Mobile-services Switching Center*) ou commutateur de service. qui gère toutes les communications avec les mobiles sous sa couverture.
- HLR (*Home Location Register*) ou base de données de localisation nominale où sont stockées toutes les informations relatives aux abonnés du PLMN.
- VLR (*Visitor Location Register*) ou base de données de localisation locale dans laquelle sont stockées les informations relatives aux utilisateurs d'une région particulière.
- Le sous-système réseau utilise deux autres bases de données, l'EIR qui contient la liste de tous les mobiles, identifiés par leur IMEI, autorisés à fonctionner dans le réseau, et l'AUC (*Authentication Center*), qui contient les codes PIM des cartes SIM.

c. **Le sous-système d'exploitation et de maintenance ou OMC** assure un monitoring distant des performances du réseau GSM et permet une re-configuration à distance ainsi qu'une activité de gestion des fautes, des alarmes et des événements survenus.

L'introduction de la transmission en mode paquet dans GSM est réalisée avec le réseau GPRS (*General Packet Radio Service*) [Seu03] qui se greffe sur le réseau GSM existant, notamment pour la partie ressources de l'interface radio. Le réseau GPRS et le réseau GSM fonctionnent en parallèle : le premier est utilisé pour le transport des données, et le second pour les services classiques de voix. Tous deux utilisent les mêmes équipements du sous-système radio. C'est ensuite qu'ils se distinguent. Le réseau cœur du GPRS est un réseau paquet interconnecté, pouvant être relié à divers types de réseaux de données fixes – IP (*Internet Protocol*) – ou encore à d'autres réseaux GPRS, exploités par d'autres opérateurs. De nouveaux équipements, protocoles, interfaces, etc. sont intégrés au réseau GSM.

2.4.3 Vulnérabilités et failles du réseau GSM/GPRS

Avant d'aborder les services de sécurité et les attaques potentielles qui ont été conduites sur le réseau GSM/GPRS, il est nécessaire de mentionner les faiblesses du réseau GSM/GPRS. Le réseau en gros, présente un nombre de faiblesses dues à sa conception. Les concepteurs ont perçu que seule la voie radio nécessite une protection supplémentaire dans le but de conserver un coût acceptable. Tandis que le minimum de protection est assuré pour les autres voies de transmission. Cette supposition et plusieurs autres ont introduit des défauts dans la sécurité du réseau qu'on peut résumer par les points suivants :

a- les concepteurs ont commis une erreur fondamentale en supposant que la partie fixe du réseau présente une sécurité acceptable. Cette supposition a rendu le niveau de sécurité du réseau GSM identique à celui du réseau fixe RTC ou RNIS. A noter que plusieurs menaces affectent le réseau fixe comme le « *wire tapping* »

b- le choix des milieux de transmission a conduit à une autre supposition erronée. Ils n'ont protégé que l'interface radio supposée la plus faible étant en milieu ouvert. Et ils ont supposé que l'utilisation des câbles enterrés assure la sécurité du reste du réseau. Ce qui n'est pas toujours le cas. Les stations de base transmettent parfois leur trafic vers le MSC par micro-onde. Le faisceau micro-onde ne peut pas être parfaitement dirigé de l'émetteur vers le récepteur. Il présente des lobes secondaires qu'un espion peut intercepter avec un récepteur bien placé. Comme les données envoyées sur cette voie ne présentent aucune sorte de protection (pas de chiffrement), l'espion peut donc extraire toutes les identités (IMSI) sur les canaux de signalisation et écouter toutes les conversations sur les canaux de trafic.

c- L'interface radio n'est pas le seul point vulnérable dans GSM. Les données ne sont chiffrées que sur cette interface, elles sont envoyées en clair à l'intérieur du réseau. Ainsi, si un attaquant peut gagner l'accès au réseau de signalisation, il peut écouter toute information, depuis la conversation en cours, jusqu'à la clé de chiffrement et autres informations importantes et peut-être l'IMSI, le TMSI et l'IMEI. Il est presque sûr que la plupart des équipements d'écoute du GSM utilisent cette vulnérabilité surtout ceux utilisés par les forces des gouvernements.

d- Les liaisons entre les parties du réseau ne sont pas protégées. Sur ces liaisons, des informations très importantes comme les identités des abonnés sont transmises. Si par exemple, un abonné change sa location, le nouveau VLR doit demander au HLR l'identité de cet abonné. Ce qui est très dangereux surtout si ce VLR est dans un autre pays (*roaming*).

e- Le réseau GSM introduit une faiblesse au niveau de la procédure d'authentification, spécifiquement dans l'identité de l'abonné (IMSI) qui devrait rester secrète. Ainsi, le système GSM utilise le TMSI pour ne pas transmettre l'IMSI sur la voie radio. Mais lorsque l'abonné met en marche son appareil, l'IMSI est transmise en clair. Un espion peut donc l'intercepter avec un équipement convenable. Un autre défaut de la procédure d'authentification est que l'IMSI et l'identité du HLR sont envoyés vers le VLR lors de l'itinérance. Donc ces deux identités deviennent connues par le VLR du pays visité.

f- Si l'on suppose que l'algorithme de chiffrement (A5) est aussi robuste que possible le niveau de sécurité de la méthode de chiffrement dépend de la longueur de la clé de chiffrement (Kc), qui est de 64 bits de longueur. Mais il a été démontré que la longueur effective est de 40 bits. Cette clé peut être trouvée en utilisant un PC capable de faire 1 millions d'opérations par seconde. Mais cette opération prendra 12,7 jours, ce qui n'est pas pratique. De même, l'A5 présente, en lui-même, des défauts : plusieurs erreurs ont été trouvées dès qu'il a été publié. Pour citer quelques attaques qui on été effectuées sur ces algorithmes, en Avril 1998, l'association des cartes à puce SDA (*The Smartcard Developer Association*) avec le département de recherche à l'université de Berkely (*U.C. Berkeley*) ont pu craqué l'algorithme COMP128 sauvegardé sur la carte SIM. Ils ont pu retirer la clé individuelle Ki dans quelques heures. Ils découvriront que la clé de chiffrement Kc utilisait 54 bits seulement au lieu de 64 bits. En Août 1999, l'algorithme de chiffrement A5/2 a été craqué en utilisant un seul ordinateur dans une période de temps ne dépassant pas les quelques secondes. En Décembre 1999, Alex Biryukov, Adi Shamir et David Wagner ont publié les méthodes pour casser l'algorithme de chiffrement A5/1. Pour une période de deux minutes d'interception d'un appel, la durée de l'attaque contre cet algorithme pour le casser a duré 1 seconde. Et en Mai 2002, le groupe de recherche IBM découvre une nouvelle façon d'extraire rapidement les clés de l'algorithme COMP128 en utilisant des canaux à part.

g- Il est connu que la sécurité d'un réseau est relative à son maillon le plus faible. Les attaques peuvent prendre origine à partir de sources externes ou bien par les partenaires mobiles globaux. Pour cela, les opérateurs doivent s'assurer de sécuriser toutes les interfaces, surtout la connexion avec les réseaux externes comme le réseau Internet. Ces attaques [Bav04] peuvent affecter les performances des réseaux ou bien exploiter des applications spécifiques comme les systèmes de facturation.

h- Les opérateurs mobiles GSM qui ajoutent les services du réseau GPRS doivent prendre des mesures sécuritaires pour protéger l'infrastructure de leur réseau et le trafic de leur clientèle, dû au manque de sécurité inhérente au GTP (*GPRS Tunneling Protocol*). L'attaque la plus commune est le déni de service. Un

attaquant qui gagne l'accès au GTP peut altérer les flux de données et permet de découvrir des informations confidentielles sur l'abonné.

2.4.4 La sécurité dans le réseau GSM/GPRS

A cause de l'utilisation du canal radioélectrique pour transporter les informations, les abonnés sont particulièrement vulnérables à la possibilité d'utilisation frauduleuse de leur compte par des personnes disposant de mobiles "pirates", qui se présentent avec l'identité d'abonnés autorisés et la possibilité de voir leurs communications écoutées lors du transit des informations sur le canal radio.

Le système GSM intègre donc des fonctions de sécurité visant à protéger à la fois les abonnés et les opérateurs. Essentiellement, il assure l'authentification d'un abonné pour protéger l'accès aux services, la confidentialité de l'IMSI, des données usagers, et des informations de signalisation.

Pour mettre en œuvre les fonctions d'authentification et de chiffrement des informations transmises sur la voie radio, GSM utilise les éléments suivants :

- des nombres aléatoires RAND,
- une clé individuelle Ki pour l'authentification et la détermination de la clé de chiffrement Kc,
- un algorithme A3 fournissant un condensât SRES à partir des arguments d'entrée RAND et de la clé Ki,
- un algorithme A8 pour la détermination de la clé Kc à partir des arguments d'entrée RAND et Ki,
- un algorithme A5 pour le chiffrement / déchiffrement des données à partir de la clé Kc.

A chaque abonné est attribué une clé Ki propre. Les algorithmes A3, A5 et A8 sont quant à eux les mêmes pour tous les abonnés d'un même réseau. Les services d'intégrité et de non rejeu ne sont pas assurés par le réseau GSM. Une non répudiation non cryptographique est similaire à celle assurée par le réseau RTC qui se base sur la fiabilité des factures de l'opérateur.

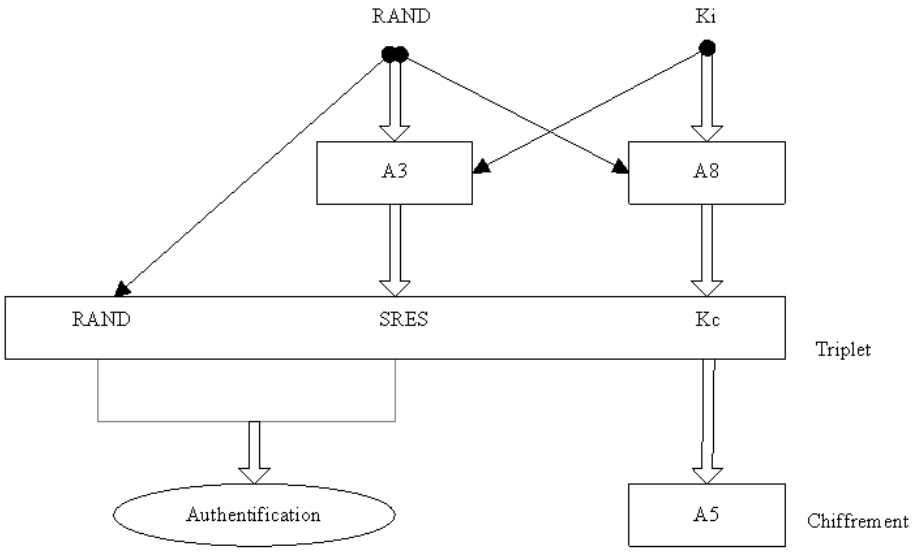

Figure 2.6 – Architecture de sécurité dans GSM

La sécurité dans le réseau GPRS (pour *General Packet Radio Service*) est très similaire au réseau GSM. Une des mesures de sécurité est l'utilisation des cartes à puce (SIM) protégée par un mot de passe (PIN). Les services de sécurité déployés dans le réseau GPRS sont :

- L'authentification : le routeur SGSN (pour *Serving GPRS Support Node*) contrôle le chiffrement et l'authentification du terminal mobile. Cette authentification est très similaire à celle offerte par le réseau GSM. Quand un abonné GPRS se connecte la première fois au réseau, le SGSN authentifie le mobile MS en utilisant la carte SIM. Il compare ces informations avec les informations d'authentification présentent dans la base de données du réseau (HLR). Durant cette phase d'authentification, un nombre aléatoire est utilisé pour générer la clé d'authentification afin de rehausser la sécurité. Cette clé n'est pas transmise sur aucune partie du réseau.
- La confidentialité : toutes les données transmises entre le mobile (MS) et le SGSN sont cryptées dans le réseau GPRS. Durant la phase d'authentification, la décision de crypter les données est prise. Le chiffrement est alors établi en générant la clé de chiffrement. L'algorithme utilisé pour le chiffrement est le GEA (*GPRS Encryption Algorithm*) qui est une version de l'algorithme A5 utilisé dans GSM.
- Le routeur GGSN (pour *Gateway GPRS Support Node*) est le point d'entrée entre le réseau Internet et le réseau GPRS. Pour sécuriser les terminaux mobiles contre les attaques en provenance de l'extérieur du réseau GPRS, des filtres de paquets (*firewalls*) GGSN sont placés pour filtrer le trafic. Le SGSN protège l'usager contre les attaques provenant des autres mobiles. Le NAT (pour *Network Address Translation*) peut aussi être utilisé pour protéger les adresses IP.
- La protection des données s'arrête au niveau du GGSN. Pour sécuriser le trafic de bout en bout, un VPN peut être créé entre le GGSN et l'intranet des entreprises privées pour assurer la sécurisation du trafic de données et de voix.

2.4.5 Conclusion

GSM est un réseau conçu à la base pour transporter la voix. Les performances sont étudiées pour ne pas affecter la qualité de la voix. Les algorithmes d'authentification et de chiffrement sont câblés dans la carte SIM (A3 et A8) et dans le téléphone mobile lui-même (A5). La carte SIM qui contient l'identité de l'abonné sert à identifier ce dernier auprès de l'opérateur et à assurer la mobilité de l'usager. Les services de sécurité couverts par GSM sont l'authentification de l'usager et le chiffrement sur le lien radio uniquement, entre l'opérateur et la station mobile, car la partie câblée de l'opérateur est considérée comme fiable. Cependant, malgré toutes les précautions prises par les constructeurs pour garder les algorithmes sûrs, des failles et des erreurs ont été découvertes dans ces algorithmes lors de leur publication et des attaques ont pu être effectuées pour les casser.

GPRS, par sa définition, offre des services de données aux usagers. Il est basé sur la technologie IP. Le service de téléphonie qu'il peut offrir est la voix sur IP sur un réseau mobile. Les attaques et les services de sécurité du réseau IP et de la voix sur IP s'applique en grande partie au service de voix sur IP, si déroulé sur le réseau GPRS et peuvent atteindre les entités du réseau GSM.

2.5 Réseaux UMTS

2.5.1 Introduction

L'UMTS [Luc00] est le réseau mobile offrant des services de multimédias. L'UMTS peut être vu comme un système Internet mobile. La technologie de l'accès radio est basée sur le W-CDMA. L'UMTS est désigné pour satisfaire les requis de l'IMT-2000 [IMT-2000], qui est le standard phare pour assurer des débits supérieurs à 2Mbps.

2.5.2 Architecture du réseau

L'UMTS est la troisième génération du réseau GSM standardisé par l'ETSI. L'UMTS est considéré comme étant la vision européenne de la 3ème génération des réseaux mobiles. L'UMTS est un réseau mobile puissant qui supporte le multimédia.

L'architecture de l'UMTS s'appuie sur la modularité. Ses éléments constituants doivent être indépendants, de façon à autoriser en théorie des mises à jour de telle ou telle partie du système sans avoir à en redéfinir la totalité.

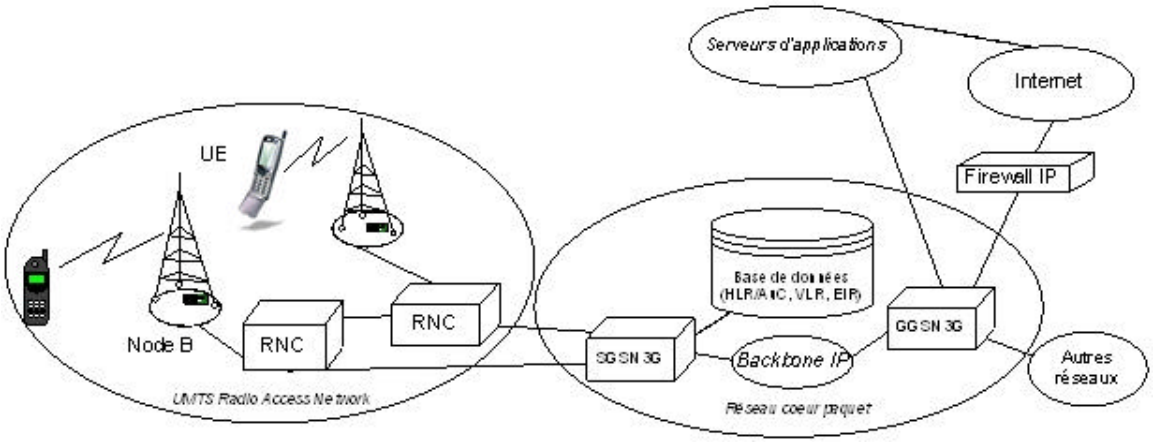

Figure 2.7 - Architecture du réseau UMTS

La figure 2.7 illustre l'architecture générale de l'UMTS. L'UTRAN regroupe les stations de base, appelées NodeB dans le vocabulaire de l'UMTS, et les contrôleurs de stations de base, ou RNC (*Radio Network Controller*).

Un RNC est l'équivalent d'un BSC dans un réseau GSM. Il contrôle les ressources radio de son domaine. Un réseau UTRAN peut inclure un ou plusieurs sous-systèmes radio RNS (R*adio Network Subsystems*) ; ce dernier comprend un ou plusieurs NodeB. Le RNC de service de l'UE gère le NodeB pour le contrôle de charge et de congestion. Le rôle principal du NodeB (équivalent à la BTS dans GSM) est de réaliser le traitement des fonctions de la couche physique. Le RNC réalise les fonctions de couche 2 (*MAC*) pour les données en provenance ou vers l'interface radio ainsi que les fonctions de gestion de mobilité.

Scindé en deux parties, le réseau cœur est constitué d'un réseau cœur de type circuit (*CS_Domain*) et d'un réseau cœur de type paquet (*PS-Domain*). Le réseau cœur circuit est composé, à l'image de celui du GSM, de commutateurs de circuits, les MSC

(*Mobile-service Switching Center*), de passerelles vers les réseaux téléphoniques publics, les G-MSC (*Gateway-MSC*) et de serveurs dédiés aux SMS, les SMS-GMSC (*SMS-Gateway MSC*). Le réseau coeur orienté paquet est semblable à celui du GPRS. Il est composé de commutateurs paquet, les SGSNs et les GGSNs, qui relient le réseau de l'opérateur au monde extérieur, qui peut être un réseau paquet public ou un réseau paquet d'un autre opérateur. Le réseau situé entre les GGSNs et les SGSNs est un réseau paquet quelconque, le plus souvent un réseau IP. Dans la version UMTS 99, l'ATM est le protocole de transport choisi pour le réseau cœur.

Pour gérer les données relatives aux utilisateurs, telles que leur position dans le réseau, leur abonnement, etc., les bases de données introduites dans le GSM sont toujours présentes dans l'UMTS. Il s'agit, entre autres, des HLR, VLR et EIR.

2.5.3 Vulnérabilités dans le réseau UMTS

Les vulnérabilités les plus connues du réseau UMTS peuvent se résumer comme suit :

- Manipulation des données transmises : des malveillants peuvent manipuler des données transmises sur toutes les interfaces du réseau UMTS.
- Manipulation des données enregistrées : des malveillants peuvent manipuler des données enregistrées dans des entités systèmes, dans des terminaux ou bien dans la carte USIM comme l'accès à l'IMEI.
- Manipulation par mascarade : des malveillants arrivent à se faire croire qu'ils sont des participants dans une communication et peuvent ainsi manipuler les données sur les interfaces. Il est possible aussi de manipuler le comportement de la carte USIM par mascarade par l'implantation d'applications malicieuses sur le terminal ou la carte USIM.
- Ecoute des trafics des usagers, de la signalisation et des données de contrôle sur l'interface radio.
- Analyse de trafic : les malveillants peuvent observer le temps, le débit, la longueur, la source ou la destination des messages sur le lien radio pour accéder aux informations confidentielles.
- Les faiblesses du mécanisme d'intégrité du réseau GSM sont :
    - Les clés utilisées entre le UE et le RNC et qui sont générées par la VLR/SGSN sont transmises en clair au RNC
    - L'intégrité des données de l'usager n'est pas assurée
- Durant la phase de signalisation, et pendant un laps de temps très court, les données de signalisation ne sont pas protégées et peuvent être exposées à des manipulations.

2.5.4 La sécurité dans le réseau UMTS

UMTS comme GSM utilise un module d'identification qui est une care à puce appelé USIM. Dans cette USIM, les applications, les certificats, les signatures numériques, les algorithmes de chiffrement et autres types de données sont sauvegardée dans la mémoire ce cette carte. Elle est conçue sur une technologie d'intégration et de microchip très avancée, elle possède en outre une capacité de mémoire, d'interfaçage et de calcul très élevé. Ces cartes peuvent utiliser une technique dite sans contact qui ne nécessite pas l'insertion et l'extraction de la carte du terminal pour effectuer les opérations requises.

Les services de sécurité offerts par le réseau UMTS sont :

- Authentification et échange de clés (AKA) : l'authentification prouve l'identité entre l'usager et le réseau, elle est divisée en deux parties :
  - Authentification de l'usager par le réseau
  - Authentification du réseau par l'usager

  La technique utilisée est appelée *one-pass authentication* qui permet de réduire les échanges de messages. Suite à cette procédure d'authentification, l'usager est sûr du réseau et vice versa. L'authentification est nécessaire aux autres mécanismes de sécurité comme la confidentialité et l'intégrité.
- La confidentialité : elle est assurée par le chiffrement des données des usagers et de signalisation entre l'usager et le réseau en utilisant l'identité temporaire (TMSI/P-TMSI) de l'usager à la place de son identité globale (IMSI). Le cryptage se fait entre la carte USIM et le RNC. La confidentialité de l'usager est assurée entre l'abonné et la VLR/SGSN. Si le réseau n'offre pas le service de confidentialité, l'usager est notifié et aura alors le choix d'accepter ou de refuser les connexions. Les éléments qui sont confidentiels sont l'identité de l'abonné, la localisation courante de l'usager, les informations (voix et données) de l'usager et les données de signalisation.
- L'intégrité : la protection de l'intégrité est de s'assurer que les messages entre l'usager et le réseau n'ont pas été manipulé, même si le message n'est pas confidentiel en soi. Ceci est assuré par un condensât ajouté à chaque message de signalisation. Pour cela, le réseau UMTS utilise une clé pré-partagée qui est enregistrée dans la carte USIM et le AuC pour générer le condensât. Au niveau du transport, l'intégrité est assurée par le calcul du CRC.

2.5.5 Conclusion

Nous avons passé en revue rapidement la sécurité dans le réseau UMTS. Nous voyons bien que tout comme GSM, la confidentialité est surtout assurée au niveau du lien radio. Ce qui différencie l'authentification de UMTS de celle de GSM est que le réseau UMTS offre l'authentification du réseau à l'usager. Il assure aussi l'intégrité des données de signalisation mais ignore celle des données (voix et donnée) de l'usager.

2.6 ToIP / VoIP

2.6.1 Introduction

La ToIP consiste en un ensemble de techniques qui, dans une entreprise ou un organisme, permettent la mise en place des services téléphoniques en utilisant les standards de la voix sur IP (VoIP) pour la transmission de messages vocaux sur des réseaux de données IP. La ToIP est gérée par une entité unique, une entreprise pour ses besoins internes ou un opérateur télécom.

Dans le cas de la VoIP, on se contente d'interconnecter des PABXs en encapsulant la voix numérisée, dans les paquets IP. Ces derniers sont ensuite véhiculés au sein du réseau de données, de manière classique, comme des paquets de données. La voix est simplement « reconstituée » lorsque les paquets arrivent chez le destinataire. Un

exemple «parlant » de VoIP se trouve dans le raccordement de deux sites via une ligne spécialisée qui transporte la voix et les données.

Quant à la ToIP, elle va plus loin que la VoIP en terme de mécanismes et d'équipement, cherchant à apporter aux utilisateurs la qualité de service (qualité de transmission, qualité de voix, disponibilité du service) et les services que ces derniers ont l'habitude de trouver du côté de la téléphonie classique (présentation du numéro, transfert d'appel, conférence, etc.) le tout sans faire appel aux PABXs traditionnels.

La VoIP est donc une composante de la ToIP et sa sécurisation est, par conséquence, la condition sine qua non de la sécurisation de la ToIP.

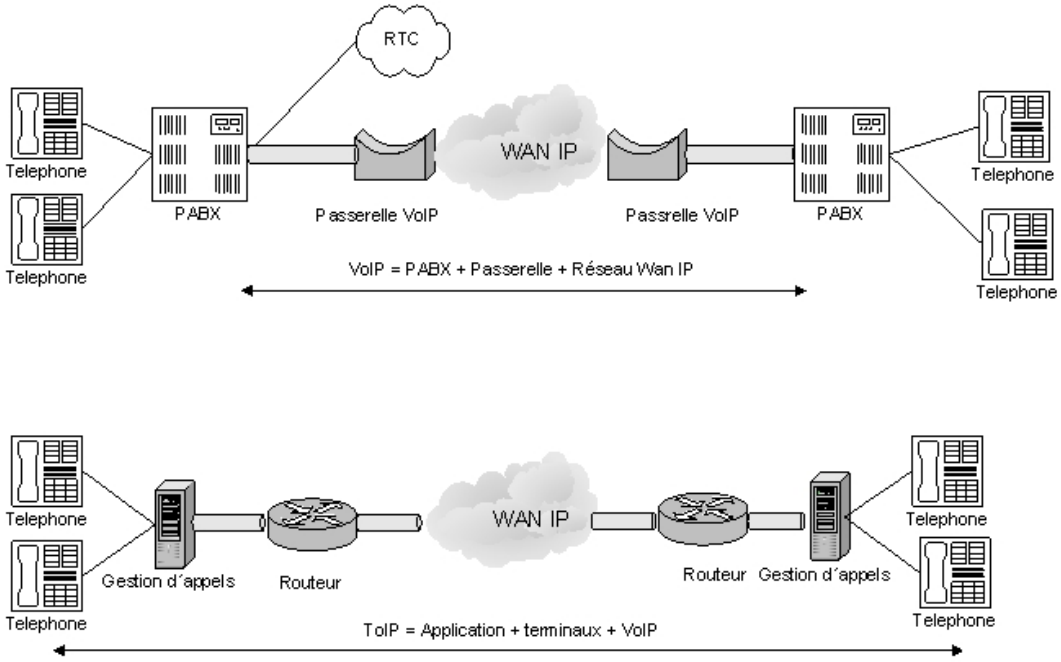

Figure 2.8 - Périmètres comparés de la VoIP et de la ToIP.

Il est aussi à noter que dans le cadre de la téléphonie sur IP, l'on distingue la téléphonie *fixe* sur IP et la téléphonie *mobile* sur IP. Cependant, nous nous limiteront dans le cadre de ce rapport à la téléphonie *fixe* sur IP, que nous désigneront tout court par ToIP, sauf indication explicite contraire.

Plusieurs standards applicatifs existent à l'heure actuelle et qui offre les services de la voix sur IP. Parmi eux, nous présenterons brièvement le standard H.323 [ITUT-H323-03] de l'UIT-T et le standard SIP [RFC2543] de l'IETF.

Dans la suite de cette section du chapitre, nous nous contenterons d'analyser l'état de l'art de la sécurité des standards de la voix sur IP et des protocoles de sécurité offerts par les couches sous-jacentes, couches de transport, de réseau et d'accès (fig.2.1). Ainsi, nous verrons la sécurité dans H.323 et SIP, puis au niveau transport, la sécurité offerte par TLS/DTLS et SRTP. Au niveau de la couche réseau, le protocole IPsec sécurise le trafic IP et enfin au niveau de l'accès, nous verrons essentiellement la sécurité apportée par les accès sans fil comme les standards WLAN et Bluetooth.

2.6.2 Architecture de la ToIP/VoIP

Depuis les années quatre-vingt, les éditeurs de logiciels cherchent à utiliser le réseau informatique pour y véhiculer de la voix. Suite à l'apparition de protocoles comme H.323 ou SIP, les constructeurs d'autocommutateurs ont progressivement intégré cette dimension IP à leurs solutions dans une optique de convergence voix-données. Dans un premier temps, cette convergence a pris la forme de cartes optionnelles à intégrer dans les commutateurs privés (PABX) existants, pour être proposée aujourd'hui de façon native. C'est ce qu'on appelle la Téléphonie sur IP (ToIP).

En offrant la possibilité de faire transiter les communications de voix sur des réseaux de données, la ToIP marque la fin d'une cohabitation entre deux réseaux distincts, très différents, du point de vue de la gestion et de la technologie. Ce transfert de l'ensemble des flux sur une infrastructure unique marque les prémices d'une convergence des réseaux multiservices existants (voix, données, vidéo).

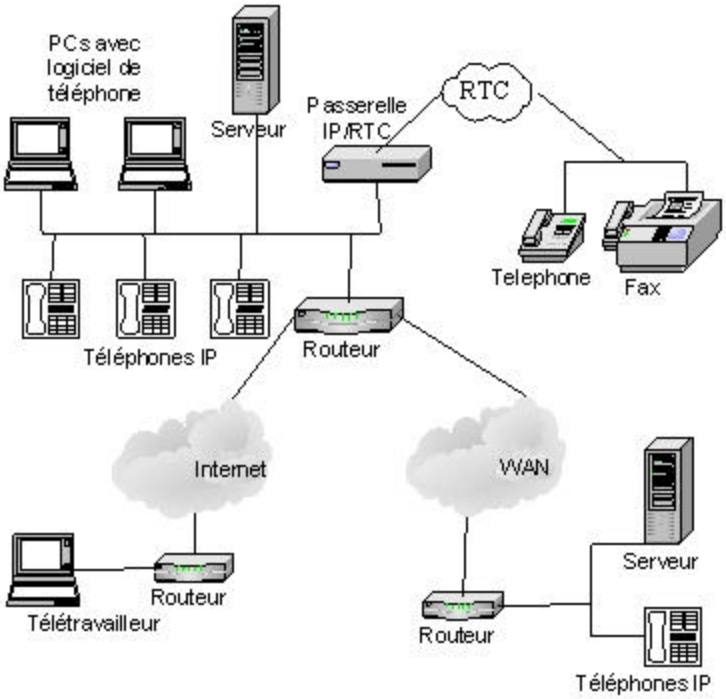

Figure 2.9 - Convergence des réseaux voix-données.

2.6.3 Vulnérabilités de la Téléphonie sur IP

Selon le *Giga Information Group [GIG]*, les sociétés qui font le choix de la téléphonie IP augmentent leurs risques d'être piratées. En effet, alors que la téléphonie traditionnelle constituait un environnement isolé, la téléphonie sur IP, avec une infrastructure commune aux données, engendre bien des interrogations concernant la sécurité.

Le serveur de communication et les terminaux IP échangent trois types de flux : le premier permet le téléchargement d'une image, qui apporte le *firmware*. Le deuxième fournit les informations de configuration, qui personnalisent le poste d'un utilisateur. Le

troisième concerne la communication téléphonique. Les trois peuvent comporter des brèches s'ils ne sont pas convenablement sécurisés.

Tous les fournisseurs de solutions de téléphonie sur IP s'accordent à dire qu'il faut absolument isoler les flux de téléphonie des flux concernant les données. Cela implique un environnement Ethernet commuté et non partagé, et la mise en place de réseaux locaux virtuels (VLAN) dont un sera dédié à la voix, avec des règles très rigoureuses pour passer d'un VLAN à l'autre. Séparer les flux vocaux des flux de données est de toute façon conseillé pour assurer une meilleure qualité de service. Pour les utilisateurs distants, la mise en place d'un réseau privé virtuel (VPN), généralement basé sur le protocole IPSec, s'impose.

Placés sur un PC, les téléphones IP logiciels « *softphones* » peuvent être particulièrement vulnérables. D'une part, le poste est exposé à des risques liés à des failles de son système d'exploitation, à des problèmes applicatifs ou à des virus. Même si aucun virus visant les réseaux de téléphonie sur IP n'a été signalé jusque là, certains experts considèrent qu'il y a un risque. En effet, le fait que les téléphones IP aient leurs propres adresses IP signifie qu'ils peuvent être infectés par des virus. D'autre part, il est exposé à toute attaque en provenance du réseau de données. C'est pourquoi, il est recommandé de protéger ses réseaux, car les attaques qui affaiblissent les réseaux, pourraient également mettre les téléphones sur IP hors service.

Cependant, des mesures de sécurité inappropriées font risquer à l'entreprise des appels longues distances non contrôlées, des conversations mises sur écoute et enregistrées à l'insu des interlocuteurs, des attaques par déni de service qui utilisent les ports de téléphonie comme points d'entrée

La menace la plus importante est le *déni de service*. En effet, une attaque de déni de service (ou DoS, pour «*Denial of Service* ») prolongée saturera ou fera tomber le PABX-IP. C'est à la fois l'attaque qui a le plus de répercussions et celle qui est la plus facile à réaliser.

À cela s'ajoutent les craintes d'utilisation frauduleuse du système, les craintes d'écoute ou d'espionnage. Le type de fraude qui, instinctivement, fait l'objet du plus grand nombre d'investigations de la part des fraudeurs est celui qui comporte un aspect lucratif, comme téléphoner aux frais de l'entreprise.

Les intrusions pourront aussi viser à écouter les communications, enregistrer les numéros appelés, subtiliser des informations (par exemple, un vol d'annuaire), ou encore modifier des informations.

Enfin, la passerelle, qui permet de connecter le réseau IP au réseau public RTC, peut être perçue comme une porte d'entrée potentielle et des applications comme les serveurs d'appels ou la téléphonie IP offrent aussi un point d'entrée sur le réseau, semblables à des « *back doors* » et, la trace de pirates entrant par cette porte dérobée pourrait se révéler difficile à suivre.

Autre aspect à examiner, les fonctions d'administration au niveau du gestionnaire d'appels. S'il comporte un serveur Web pour la gestion à distance, il faut y accéder par l'intermédiaire d'un protocole sécurisé tel que HTTPS.

2.6.4 La sécurité et la voix sur IP

2.7.1.1 Le standard H.323 et sa sécurité

La figure 2.10 montre les différents protocoles de sécurité utilisés pour transporter et sécuriser une communication VoIP basée sur les normes de la famille H.323. H.235 définit les recommandations de sécurité dans un environnement H.323 où TLS est utilisé pour sécuriser le canal de signalisation. Le trafic de voix se sert du transport RTP pour des communications de bout en bout entre des points terminaux. Les trafics de signalisation et de voix peuvent être sécurisés respectivement par TLS, et DTLS ou SRTP. IPsec permet de sécuriser tout le trafic IP, et donc aussi bien le trafic de signalisation et le trafic de voix.

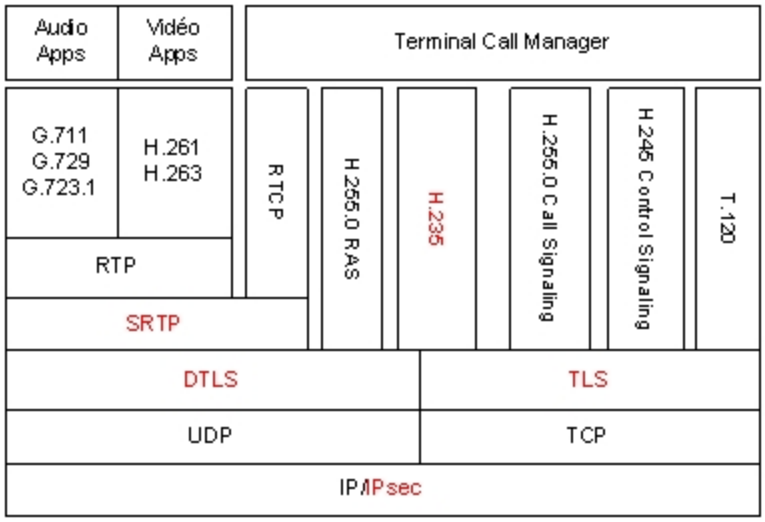

Figure 2.10 - La pile H.323 avec les différents protocoles de sécurité.

2.6.4.1.1 La sécurité avec H.235

H.235 [ITUT-H.235-03] est la recommandation de sécurité du standard H.323, préparé par le groupe d'études numéro 16 de l'ITU-T. Son but est de fournir un support pour les fonctionnalités essentielles de sécurité dans les communications H.323, comme l'*authentification*, la *confidentialité*, l'*intégrité* et parfois la *non répudiation*, dépendamment du profil de sécurité utilisé.

En effet, la recommandation H.235 propose en annexe des profils de sécurité qui utilisent les champs de H.235 pour fournir des services de sécurité au trafic H.323 en se basant sur des clés symétriques, sur des signatures digitales, ou sur des PKI [Aut02].

Il est à noter qu'en plus de la protection du trafic de voix lui-même, H.235 assure la protection de la signalisation d'appel H.225, du contrôle d'appel H.245 et de la signalisation RAS.

Cependant, cette recommandation n'est pas un standard de sécurité en elle-même. Pour assurer des mécanismes de sécurité pour la voix et pour la signalisation, elle repose sur des solutions de sécurité déjà existantes, comme IPSec ou TLS et de plus, elle ne traite pas tous les services de la sécurité.

2.7.1.2 Le protocole SIP et sa sécurité

SIP est un protocole de voix sur IP standardisé par l'IETF. Le protocole SIP suit le principe de la téléphonie. Il présente un flux de messages de signalisation et un flux de média. Pour sécuriser le flux média, SIP utilise le protocole SDP [RFC2327] (fig.2.11) pour acheminer les paramètres de sécurité nécessaires au chiffrement, comme l'indication de l'algorithme de chiffrement ainsi que la clé.

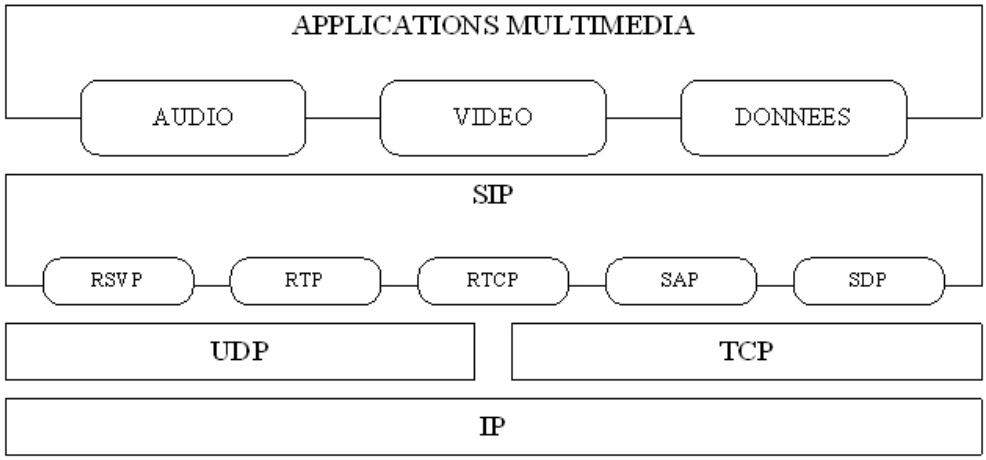

Figure 2.11 – Pile protocolaire SIP

Par contre, pour la sécurisation des échanges de messages, SIP offre des méthodes d'authentification ainsi que leur chiffrement comme suit:

- L'utilisation d'un proxy qui pourra authentifier une requête SIP. Le client doit alors fournir les données d'authentification demandées par le proxy [Lee04].
- Méthode d'authentification par code de hachage (*HTTP digest*) : les requêtes SIP, tout comme les réponses, peuvent être authentifiés par une signature électronique. De nombreux clients supportent la méthode dite « *HTTP digest* ». Cette signature électronique permet à la fois de protéger des parties en clair du message, comme des parties éventuellement chiffrées.
- Des groupes de travail à l'IETF proposent des mécanismes pour assurer l'authentification dans SIP en utilisant le protocole SAML [SAMLdraft] ou en utilisant Kerberos.
- Pour protéger les clés de chiffrement transporter dans des messages SDP et qui seront utilisés pour chiffrer le média, les requêtes SDP et les réponses doivent elles-mêmes être chiffrées en utilisant IPsec ou bien par TLS. SIP utilise aussi des mécanismes de sécurité de bout en bout qui se basent sur un secret partagé, ou sur des mécanismes à clé publique. Dans le cas d'un secret partagé, le récepteur d'un message peut le décoder à l'aide du mot de passe commun. Dans le cas d'un mécanisme à clé publique [RFC2633], l'émetteur chiffre le message avec la clé publique du récepteur. Ce chiffrement peut être effectué par l'émetteur d'une requête ou via un *proxy* dédié à la sécurité.

2.7.1.3 Les protocoles TLS/DTLS

2.6.4.3.1 La sécurité avec TLS

Le protocole TLS [RFC2246], qui est une norme de l'IETF, est le protocole le plus largement déployé pour sécuriser le trafic des applications du réseau IP nécessitant un transport fiable comme le trafic Web (HTTP) et la messagerie électronique (IMAP et POP). TLS fournit des facilités pour l'authentification, l'intégrité, la confidentialité, le non-rejeu et la non-répudiation entre les entités communicantes. Il s'insère entre la couche application et la couche réseau, pour sécuriser le trafic TCP tout en étant indépendant du protocole d'application offrant ainsi un canal sécurisé transparent.

Ainsi TLS ne fonctionne pas au-dessus de UDP. En effet, la non fiabilité du transport a une influence directe sur la conception de TLS. TLS ne permet pas un déchiffrement indépendant. C'est à dire, nous ne pouvons pas déchiffrer le *record* N si le *record* N-1 est perdu. De l'autre côté, les messages d'établissement de session (ou *Handshake)* sont supposés être bien délivrés de la source vers la destination et vice versa sinon la phase de *handshake* est interrompue. Ces deux points résument les faiblesses dans TLS.

L'implication évidente pour la téléphonie sur IP est la sécurisation du trafic de signalisation basé sur TCP. Mais la signalisation et les flots de médias qui utilisent UDP comme protocole de transport ne le seront pas.

Comme toutes les applications logicielles, la mise en oeuvre du protocole TLS peut présenter des vulnérabilités permettant à un utilisateur mal intentionné d'exécuter du code arbitraire à distance ou de provoquer un déni de service. Il est dès lors indispensable d'appliquer les correctifs correspondants aux mises à jour de ces applications.

Quelques attaques sur le protocole TLS sont théoriquement possibles. Nous pouvons diviser ces attaques en deux parties : les attaques sur les algorithmes cryptographiques utilisés par TLS et les attaques sur les réseaux et les architectures.

En dehors des attaques sur la mise en oeuvre des protocoles, l'attaque la plus répandue sur TLS est l'attaque de type MitM (pour *Man in the Middle Attack*) encore appelée "attaque de l'intercepteur". Cette attaque consiste à intercepter le trafic entre deux parties avant qu'elles ne débutent une session TLS. L'intercepteur négocie alors une session avec chaque partie et fait suivre le trafic en le déchiffrant et le chiffrant à la volée.

Des contributions et des extensions à TLS ont été introduites pour améliorer la robustesse des services de sécurité et palier aux failles de TLS.

2.6.4.3.2 La sécurité avec DTLS

Quand TLS a été développé, la limitation due à la nécessité d'utiliser un support fiable n'a pas été considérée sérieusement parce que la grande majorité des applications fonctionnaient sur TCP. Ceci est toujours vrai de nos jours, mais la situation est en cours de changement. Au cours de ces dernières années un nombre croissant de protocoles applicatifs, comme le Session Initiation Protocol [RFC2543], Real Time Protocol [RFC1889], le Media Gateway Control Protocol [RFC2705], et une variété de protocoles ont été conçues pour employer le transport sur UDP.

La solution de sécurité a été de concevoir un protocole de sécurité de canal générique qui suffira pour les applications les plus utilisées sur le transport datagramme UDP tel que TLS a fait pour TCP. Tel protocole pourrait être mis en application dans

l'espace utilisateur pour la portabilité et la facilité d'installation mais doit être assez flexible et générique pour fournir la sécurité à une variété d'applications orientées sur datagramme. Malgré de nombreux soucis que cette solution serait un projet de conception difficile, DTLS (Datagram TLS) est sorti pour qu'il puisse fonctionner correctement au-dessus du transport en datagramme. C'est une version modifiée de TLS qui prend comme référence le protocole IPsec. Cette approche a deux avantages principaux par rapport aux solutions alternatives. D'abord, puisque DTLS est très semblable à TLS, l'infrastructure préexistence de ce dernier peut être réutilisée. Pour prouver ceci, DTLS a été mis en application en l'ajoutant aux bibliothèques d'OpenSSL [OpenSSL]. En second lieu, puisque DTLS fournit une interface familière à une couche générique de sécurité, il est facile d'adapter les protocoles pour l'employer. L'expérience avec TLS a prouvé que cette facilité d'adaptation et d'implémentation est une clef au déploiement mondiale.

Vu la complexité des protocoles de sécurité modernes et de l'état actuel des techniques de protection, il est rarement possible de prouver complètement la sécurité d'un protocole sans faire au moins quelques prétentions peu réalistes au sujet du modèle d'attaque. Au lieu d'essayer de prouver rigoureusement la sécurité de DTLS, un des principaux buts dans la conception de DTLS est d'avoir des spécifications et caractéristiques de sécurité les plus proches de TLS. En conséquence, DTLS n'offre aucune amélioration à TLS. Toutes les caractéristiques présentées dans DTLS ont comme but unique le traitement sur un canal datagramme de transport non fiable, comme les temporisateurs et la retransmission.

2.7.1.4 Le protocole SRTP

Secure Real-time Transport Protocol [RFC3711] est un nouveau mécanisme de sécurité considéré pour sécuriser les réseaux de voip. SRTP est un profil de sécurité pour assurer la sécurisation des paquets RTP qui transportent les échantillons de parole des usagers. Le but de SRTP est d'assurer la confidentialité des champs utiles des paquets RTP et RTCP, l'intégrité de tout le paquet RTP et RTCP avec protection contre le rejeu ainsi que l'authentification de l'origine des messages. Ces services de sécurité sont optionnels et mutuellement indépendants. Seule la protection de l'intégrité des paquets RTCP est obligatoire pour éviter la perturbation du flux RTP. SRTP est indépendant des couches sous-jacentes utilisées par RTP. Il est caractérisé par un débit élevé et une faible expansion des paquets. SRTP utilise le cryptage symétrique, une fonction de hachage universelle pour l'authentification des messages et un numéro de séquence implicite pour le séquencement basé sur le numéro de séquence des en-têtes du paquet RTP.
SRTP est implémenté entre l'application RTP et la couche transport. Il intercepte le paquet RTP à l'entrée et envoie le paquet SRTP équivalent en sortie coté émetteur, l'opération inverse est effectué du coté récepteur.

Il est à noter que dans SRTP, les entêtes RTP sont envoyées en clair pour permettre la compression d'entête. Ce qui rend certains champs disponibles aux attaques (e.g. les numéros de séquence des paquets, les ports source et destination, et l'horodatage). En plus, il n'assure pas la sécurisation des paquets de signalisation de la voix et requiert un protocole séparé pour la génération de clés de session. Dans SRTP, l'origine des données est authentifiée, donc la source des données est authentifiée dans le cas d'un appel point à point. Mais elle n'est pas assurée dans un contexte multicast ou dans des sessions RTP groupées où l'origine des données est identifiée comme membre

d'un groupe uniquement. Nous ne pouvons dire que la personne ou même l'application est identifiée. L'origine des données peut être identifiée par l'utilisation de signature digitale pour éviter qu'une personne ne vole l'identité d'une autre personne (*spoofing*). Mais elle a un impact négatif sur la bande passante et le temps de traitement.

2.7.1.5 Le protocole IPsec

Le protocole IPSec est une suite de protocoles désignés pour sécuriser les communications au niveau de la couche réseau. La suite de protocoles est constamment en évolution depuis 1995. De nouveaux drafts sont proposés au sein du groupe de travail à l'IETF. IPSec propose deux protocoles de sécurité du trafic IP : *Authentication Header* (*AH*) [RFC 2402] et *Encapsulating Security Payload* (*ESP*) [RFC 2406]. Les services de sécurité offerts par IPSec sont:

- L'authentification des extrémités: cette authentification mutuelle permet à chacun de s'assurer de l'identité de son interlocuteur. Rappelons tout de même qu'IPSec est un protocole de niveau 3 et qu'il ne fournit donc qu'une authentification des machines mettant en oeuvre le protocole plutôt que des personnes utilisant réellement la machine. Ce n'est pas la personne elle-même qui est authentifiée, sauf si des certificats sont utilisés.
- La confidentialité des données échangées: IPSec permet, si on le désire, de chiffrer les paquets IP pour éviter que quiconque ne les lise. IPsec offre une protection contre les écoutes et analyses de trafic; il permet de chiffrer les adresses IP réelles de la source et de la destination, ainsi que tout l'en-tête IP correspondant.
- L'intégrité des données échangées: IPSec permet de s'assurer qu'aucun paquet n'a subit de modification quelconque (attaque dite active) durant son trajet par utilisation d'une fonction unidirectionnelle de hachage.
- La protection contre le rejeu: IPSec permet de se prémunir contre les attaques consistant à capturer un ou plusieurs paquets dans le but de les envoyer à nouveau (sans pour autant les avoir déchiffré) pour bénéficier des même avantages que l'envoyeur initial.

Il est à signaler qu'une implémentation IPsec gère une base de données des associations de sécurité (SAs). Une association de sécurité détient la façon de traiter un paquet IP, elle est unidirectionnelle. Pour assurer la sécurité de la communication entre deux entités, deux associations de sécurité seront négociées. Les informations incluent des paramètres tels la transformée de cryptage et les clés, etc.

Par ailleurs, deux facteurs principaux affectent la transmission de la voix quand IPSec [VoIPsec] est utilisé :

- l'augmentation de la taille des paquets, essentiellement causée par les en-têtes ESP et le nouvel en-tête IP nécessaire pour le tunnel ;
- le temps nécessaire pour crypter les en-têtes et les données utiles et la construction de nouveaux paquets.

2.7.1.6 Réseau sans fil WiFi

Les réseaux 802.11 [IEEE 802.11] forment le réseau d'accès sans fils à des réseaux Ethernet à commutation de paquets. Comme les réseaux Wireless dont nous

avons étudié l'aspect sécurité (GSM), les réseaux 802.11 ont une interface radio à sécuriser.

La norme 802.11 utilise WEP comme protocole de chiffrement symétrique afin de protéger l'interface radio. Le protocole WEP [Mil02] présente des failles importantes et n'accomplit pas correctement tous ses engagements de sécurité. De ce fait, la norme 802.11b a annoncé une version évoluée du WEP qui apporte des améliorations en terme de sécurité, mais ne résout pas tous les problèmes de sécurité posés par WEP. En effet, la grande faiblesse de WEP provient de l'utilisation de clés statiques RC4 et de l'absence de la distribution dynamique de clés de session et du mécanisme d'authentification mutuelle.

Puisque WEP présente des failles importantes et par sa définition n'offre pas une gestion dynamique de clés, la norme 802.1X [802.1Xdraft] a été introduite. Elle s'appuie sur trois entités (terminal, AP et serveur). Elle gère l'authentification via un serveur centralisé. Elle définit plusieurs techniques d'encapsulation utilisées dans le but de transporter les paquets d'authentification et de gestion du protocole EAP [RFC2284]. Le protocole EAP est utilisé avec 802.1X d'une manière transparente entre le mobile et le serveur d'authentification au travers du point d'accès (AP). Cependant, 802.1X nécessite la coopération entre un serveur (comme RADIUS) et une méthode d'authentification. La méthode est une couche au-dessus de EAP, elle définit des mécanismes de sécurité et de distribution de clés. Néanmoins, 802.11 n'a pas précisé la façon d'implémenter EAP avec 802.1X. Pour cette raison, plusieurs couches sont définies au-dessus de EAP, citons EAP-TLS [RFC2716] et EAP-TTLS [EAP-TTLSdraft].

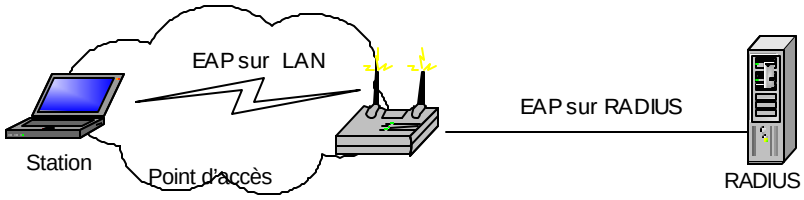

Figure 2.12 – EAP et 802.11

Cependant, ces méthodes ne sont pas toujours protégées contre l'attaque de l'homme du milieu (*Man-In-The-Middle attack).* En effet, elles ne relient pas cryptographiquement les clés de session dérivées par le protocole interne avec celles dérivées par le protocole externe. Avec ces méthodes, le nombre de messages d'aller-retour augmente et le client ne peut pas vérifier le certificat du serveur avant la terminaison avec succès de la phase d'authentification.

Une solution est proposée pour utiliser la carte à puce EAP dans la sécurité du réseau Wi-Fi. Elle supporte plusieurs méthodes d'authentification. La majorité des méthodes d'authentification EAP ne fournissent pas tous les services de sécurité; notamment l'échange anonyme de données et la protection d'identité du client. En plus, la plupart des méthodes qui sont construites pour répondre à l'anonymat sont victimes de l'attaque *Man-In-The-Middle*. Ces méthodes utilisent les PKIs qui sont très coûteuse en terme de déploiement, de calcul et de charge protocolaire.

Pour cela, le protocole EAP-Double-TLS [EAP-Double-TLSdraft] a été nouvellement défini. Il se compose de deux phases établies entre la station mobile et le serveur d'authentification.
Premièrement une session TLS-PSK permet l'authentification mutuelle et la génération des clés. Cette phase n'exige pas de certificat et établit un canal chiffré. Dans la deuxième phase, les certificats peuvent être échangés. Cette phase consiste à établir une session TLS anonyme et permet de modifier les identificateurs des sessions successives et rend anonymes les crédits utilisés lors de l'authentification. Ce protocole satisfait les recommandations de IETF-EAP; notamment l'authentification mutuelle, la génération de clés et la confidentialité de messages de notification. Ce protocole fournit la protection contre l'attaque MitM retrouvée dans EAP-TTLS et PEAP. Cela est assuré grâce à l'authentification mutuelle de la première phase de Double-TLS. Il protège contre les attaques passives et actives qui peuvent compromettre les clés dérivées du *master_secret*.

Finalement, le groupe de travail 802.11i [802.11idraft] est formé au sein de l'IEEE 802.11 dans le but d'étudier une architecture destinée à combler les lacunes de WEP et de spécifier d'une manière détaillée une méthode de distribution des clés. Ce groupe a défini une solution intermédiaire appelée WPA45 [Mil02]. Elle regroupe le 802.1X (pour l'authentification mutuelle) et le protocole TKIP46 (pour le chiffrement avec une clé dynamique). Ce standard s'appuie sur les réseaux sans fil 802.11 et utilise 802.1X pour l'authentification et le calcul d'une clé maître nommée PMK. Cependant, dans le cas du mode ad-hoc, cette clé baptisée PSK est distribuée manuellement. La hiérarchie des clés cryptographiques est définie par [802.11idraft].

2.7.1.7 Réseau personnel sans fil Bluetooth

La technologie Bluetooth [Mor02] permet principalement les échanges de données entre équipements électroniques sur des distances plus courtes que pour les WLAN, pour cela, les réseaux Bluetooth sont appelés PAN (*Personal Area Networks*). Il permet la transmission de la voix et de données. Bluetooth étant issu du monde de la téléphonie, son mode de sécurisation a été calqué sur celui des téléphones sans fils numériques. Il s'agit d'une sécurisation point à point par échange de clés privées.

L'authentification sous Bluetooth utilise la stratégie de défi/réponse. Le protocole d'authentification utilise une clé symétrique. Alors une authentification réussie est basée sur la même clé possédée par les deux partis. Les deux participants utilisent la fonction d'authentification E1 avec un nombre aléatoire, l'adresse Bluetooth et la clé de lien courante pour calculer la réponse au défi lancé. La réponse est envoyée à l'autre parti qui à son tour vérifie l'identité. Cependant, pour assurer une authentification plusieurs attaques peuvent être conduites, essentiellement à partir du code PIN, des différentes clés dont la robustesse est liée essentiellement à la longueur de la clé et de l'adresse Bluetooth de l'équipement par viol d'identité et donc d'adresse. Il est à noter que c'est l'équipement qui est identifié et non pas l'utilisateur.

Le système de chiffrement dans Bluetooth chiffre les données utiles (payload seulement) et non l'ensemble des paquets. Ceci est fait par l'algorithme de chiffrement E0 qui est re-synchronisé avec chaque bloc de données utiles.

Comme la taille de la clé de chiffrement varie de 8 à 128 bits, cette taille doit être négociée entre deux équipements désirant communiquer. Dans chacun d'eux, il y a un paramètre définissant la taille maximale permise pour une clé.

La procédure de chiffrement dans Bluetooth présente plusieurs vulnérabilités. L'algorithme de chiffrement E0 avec une clé de 128 bits peut être cassé dans certaines circonstances. Nous n'allons pas nous attarder sur la preuve mathématique dans ce document. Toutefois, cette attaque est prise en compte dans la spécification Bluetooth. Pour l'exécuter on a besoin d'accéder au KeyStream chiffrant chaque paquet. Or le KeyStream a une fréquence de re-synchronisation assez grande de sorte à ce qu'un bout de keyStream différent est utilisé pour chiffrer chaque trame. Alors la possibilité d'attaque devient médiocre.

2.6.5 : Une application de voip et sa sécurité

Skype [skype] est une application voip cliente se basant sur la technologie *peer-to-peer* [And04]. Elle n'utilise ni H.323 ni SIP avec un serveur central par lequel passe les appels, mais plutôt chaque client Skype [Bas04] recherche d'autres clients Skype d'une façon décentralisée. De ces connections, ce client construira alors un réseau qui l'aidera à rechercher d'autres clients Skype et leur transmettra des messages textuels. Les clients peuvent être des nœuds simples ou bien des super-nœuds. Les noeuds simples se connectent à un super-nœud et doivent s'enregistrer avec le serveur d'authentification pour pouvoir se loguer sur le réseau. Les super-nœuds ont pour rôle de router les appels vers le client destinataire. Cette application est très populaire dans le monde la voip, sans frais, et assure des appels sans frais aussi sauf pour passer au réseau RTC. C'est une application simple à déployer à comparer avec les autres plates-formes voip. En plus de la voip, elle offre des applications de messagerie instantanée, de moteur de recherche et de transfert de fichier. A la différence d'autres applications voip qui se base sur SIP comme le réseau Vonage, les clients Skype peuvent fonctionner avec les filtres firewalls et la translation d'adresse (NAT). Pour cela, Skype utilise une version modifiée des protocoles STUN [RFC3489] et TURN [TURNdraft] pour pouvoir déterminer le type de NAT et de *firewall* derrière lesquels il se trouve.

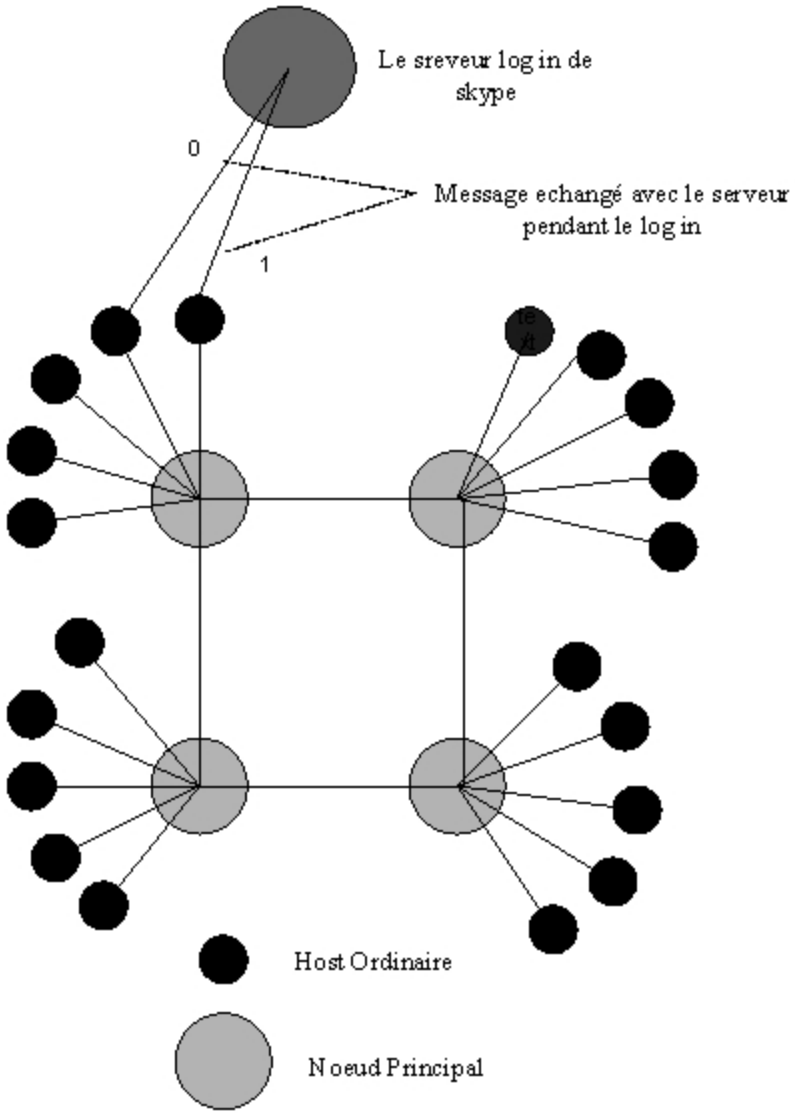

Figure 2.13 – le réseau Skype et ses différents éléments

Les services de sécurité assurés par Skype [Sim05] sont:

- L'authentification: Skype utilise un serveur central d'authentification pour authentifier ses utilisateurs. Chaque utilisateur de Skype possède un identifiant et un mot de passe qu'il utilise pour se loguer au réseau. Le mot de passe est généré aléatoirement pour chaque utilisateur et est envoyé par email à l'utilisateur en question par le serveur d'authentification. Selon les publicités de Skype, chaque identifiant et la distribution électronique de l'application Skype sont signés numériquement par une clé privée RSA. La clé publique correspondante est incluse dans chaque exécutable de Skype. Ainsi on pourra dire que chaque usager possède un certificat public.
- La confidentialité des conversations: Skype utilise l'algorithme RSA à 1536 ou 2048 bits pour négocier les clés symétriques de chiffrement et l'algorithme AES à 256-bit pour le chiffrement. Les clés publiques des usagers sont certifiées par le serveur d'authentification lors de la phase d'authentification de l'usager. Les recherches

[Sim05] ont montré que les communications nécessaires avant d'établir une communication ne sont pas cryptés et donc accessible à des participants non privilégiés. Ainsi, ces communications peuvent être sujettes à des attaques d'analyse de trafic même si on n'arrive pas à reproduire les conversations à cause du cryptage.

- L'intégrité des conversations: les provisions d'intégrité de Skype sont complètement inconnues.

Cependant, malgré les fonctions de sécurité que Skype prétend assurer, la société qui a créé Skype n'a pas publié l'algorithme d'échange de clé, ni le protocole, ni les certificats ni son système d'authentification et finalement, même l'implémentation de chiffrement n'est pas publiée. On reviendrait au même cas que le réseau GSM où les algorithmes d'authentification et de chiffrement n'étaient pas publiés et avec le temps il s'est avéré qu'ils sont pleins de bugs et leur sécurité est facilement cassable. Plusieurs scénarios d'attaques contre la sécurité offerte par Skype sont décrits dans [Sim05].

2.6.6 Conclusion

Nous remarquons d'après tout ce qui précède que la sécurité totale de la communication VoIP par un seul protocole reste impossible, au niveau application la sécurité assurée par SRTP reste incomplète et la sécurité au niveau IPSec malgré qu'elle est plus complète que celle de SRTP (protection des adresses source et destination et le cryptage de toutes les données utiles relatives au trafic VoIP). Cependant IPsec entraîne un grand *overhead*, ce qui affecte la qualité de la voix sur IP.

TLS est de plus en plus le protocole phare de la sécurisation des échanges dans le monde Internet et sans fil. En effet, une version future de TLS saura répondre aux différents inconvénients et limitations lorsque TLS sera appliqué comme protocole d'authentification et de confidentialité.

DTLS est une version modifiée de TLS pour qu'il puisse fonctionner correctement au-dessus du transport en datagramme, il offre les mêmes services de sécurité que TLS et présente des performances similaires à TLS. En raison de la simplicité et de la facilité de déploiement, DTLS fournit une alternative attirante à la sécurité des applications de téléphonie sur IP.

Les réseaux WLAN sont initialement conçus pour transporter des données. Mais vu l'exigence actuelle de mise en place de la mobilité et des applications de voix et de multimédias à contraintes temps réel, le standard 802.11 a subi plusieurs modifications pour assurer la sécurité des transferts sur les liens radio, et la qualité de service des applications sensibles. Quant aux réseaux Bluetooth, ils sont initialement conçus pour supporter le trafic à contraintes temporelles. Des failles de sécurité existent aussi dans Bluetooth, et sont dues à l'algorithme E0 qui peut dans certains cas être cassé [casser E0]. Certains services de sécurité ne sont pas assurés par Bluetooth comme l'intégrité, le non rejeu et la non-répudiation.

2.7 Analyse et comparaison des architectures de sécurité existantes

Le délai maximum acceptable pour une transmission optimale de la voix est de 150 ms [ITU G.114] pour les communications sur voies bifilaires et 250 à 300 ms pour les communications satellitaires. Introduire des fonctions sécuritaires à la voix

garantissant la confidentialité, l'intégrité et l'authentification affecte la qualité de la voix et augmente le délai de transmission et le temps de traitement.

Dans un réseau traditionnel, seul le numéro de la ligne (*Caller ID*) peut être authentifié, pas la personne. Le *Caller ID* est acheminé dans des messages de signalisation entre l'appareil téléphonique et le commutateur auquel est rattaché l'abonné. Ce *Caller ID*, à la demande de l'abonné peut être envoyé vers l'abonné appelé. Le réseau sémaphore qui transporte la signalisation ne déploie pas des moyens de sécurité. De plus, de nos jours, le réseau sémaphore est connecté au réseau Internet qui est un réseau ouvert et sujet à des attaques de toutes sortes. Donc, il est possible de voler l'identité d'une autre personne lors de l'établissement d'un appel. La confidentialité est assurée par l'utilisation d'équipements spécifiques qui chiffrent les données émises.

Dans un réseau GSM, les fonctions de sécurité décrites ci-dessus sont appliquées uniquement sur la liaison radio entre la station mobile et le point d'accès au réseau. Les communications et la signalisation dans la partie fixe du réseau ne sont pas protégées. Aucun algorithme n'est fourni pour assurer l'intégrité des données acheminées. Dans le réseau mobile, à chaque carte (U)SIM est attribué une identification unique du détenteur de la carte qui ne circule pas sur le lien radio mais est enregistrée dans la base de donnée et dans la carte (U)SIM. A chaque nouvel appel, un identificateur temporaire est utilisé. Mais là aussi, c'est la carte SIM qui est identifiée et non pas l'abonné. Si la carte est volée, une autre personne peut s'identifier comme étant la personne initiale. Cette authentification se fait uniquement entre l'abonné et le réseau. Tout comme le réseau téléphonique classique, l'identifiant du numéro de l'abonné mobile, le *Caller ID*, est envoyé vers l'abonné appelé comme une preuve d'identification. La confidentialité est assurée par le chiffrement des données transmises sur le lien radio.

Bluetooth assure l'authentification des équipements uniquement. Il n'y a pas moyen d'authentifier l'utilisateur d'un équipement Bluetooth. L'intégrité et la non-répudiation ne sont pas assurées. Les données échangées ne sont pas protégées contre les attaques de rejeu. La sécurité est négociée lors de l'établissement d'une connexion, la sécurité n'est plus négociée au besoin lors des échanges d'information dans une connexion déjà établie. Dans ce cas-là, toute application devrait implémenter la sécurité au niveau des couches supérieures pour assurer l'authentification de l'utilisateur et le contrôle d'accès à la base de données. Le service audio offert par Bluetooth utilise un canal spécifique et ne passe pas par les différentes couches de la pile protocolaire Bluetooth ou bien un service de voix sur IP peut passer à travers toutes les couches protocolaires de Bluetooth.

La voix sur Wi-Fi basé sur les normes IEEE 802.11 est naturellement acheminée dans des paquets IP utilisant les protocoles temps réel (e.g. SIP, H.323, MEGACO). Un problème de performances s'impose si les couches MAC et Physique de 802.11 sont utilisées. Typiquement, VoIP utilise UDP même sur des réseaux de type Best Effort et la correction d'erreurs et la retransmission doivent être faites aux niveaux supérieurs car pour le trafic temps réel, les couches basses doivent réduire le temps de latence et la gigue autant que possible. L'introduction de tels mécanismes au niveau bas augmenterait

les délais. Vu que les pertes et les erreurs sont beaucoup plus fréquentes sur le lien radio que dans un réseau câblé, la couche MAC de 802.11 introduit un mécanisme d'acquittement qui introduit de la fiabilité au niveau du lien radio. Mais dans le cas des applications de voix, ce protocole rajoute de la gigue et de la latence au trafic de voix. D'où l'idée d'améliorer la couche MAC pour mieux faire passer la voix dans la version IEEE 802.11e [IEEE 802.11e]. WLAN offre une multitude de solutions de sécurité au niveau de l'accès, qui pour la plupart sont en cours de standardisation à l'IETF. Essentiellement, ces solutions sécurisent le trafic de données. Des études devraient être faites pour pouvoir connaître leur impact sur la qualité de la voix sur IP, bien que WLAN offre un nouveau standard pour assurer une meilleure qualité de service pour la voix sur IP sur WLAN (ou Wi-Fi). Ces études n'entrent pas dans le cadre de cette thèse, mais nous pouvons les citer comme des perspectives pour la continuité de travail.

La notion de signalisation de l'appel et d'échanges des informations de voix existent toujours même en passant à la voix sur IP. Par ailleurs, toutes les contraintes de la VoIP qui seront citées ultérieurement sont inhérentes à la ToIP et les contraintes et défauts inhérents à IP sont les fondements des difficultés rencontrées par le concept VoIP. Or, le transport de la ToIP ne doit souffrir d'aucun retard de transmission, ni d'altérations dues aux *firewalls*, ni de perte de paquets. Due à ces contraintes temporelles, la taille des paquets de voix se limitera entre 10 et 50 bytes de données utiles.

De plus, les applications temps réel – i.e., la vidéoconférence, Voix sur IP, vidéo temps réel – imposent des contraintes temporelles, présentent certaines vulnérabilités au niveau de la transmission des paquets et posent des problèmes de synchronisation temporelle. Le transfert de la voix sur un réseau IP est soumis à trois principales contraintes et qui sont:

- Le délai de latence : Pour une application voix sur IP, plusieurs facteurs influencent le délai durant une transmission, à savoir les délais d'émission des paquets, les délais de propagation sur les lignes de communications, les délais dans les mémoires tampons des équipements de commutation et de routage, etc. Les délais d'acheminement du signal vocal sont d'autant plus importants que les délais dus à l'écho le sont. Tant que l'écho est inférieur à 50 ms, il n'est pas perceptible. Plus il est décalé dans le temps, plus il est insupportable. La valeur optimale couramment admise en VoIP, pour le délai d'un aller simple, est inférieure ou égale à 100 ms et ce délai reste acceptable jusqu'à une valeur de 200ms afin de respecter les contraintes d'une conversation interactive. La sécurisation de la voix sur IP augmente considérablement le délai de latence du fait des opérations cryptographiques introduites par le déploiement des différents services de sécurité. Introduire un haut niveau de sécurité nécessite une bonne étude des paramétrages et un juste compromis entre une haute QoS et le niveau de sécurité requis.
- La gigue (ou *Jitter*) : La gigue représente la variation des écarts de délais de transmission entre des paquets consécutifs. Elle nécessite la mise en place de mémoires tampons en réception qui lissent ces écarts pour retrouver le rythme de l'émission, ce qui a pour effet néfaste d'augmenter le délai de transmission. La valeur optimale couramment admise en VoIP, pour la gigue, est inférieure ou égale à 40 ms et la valeur acceptable est inférieure ou égale à 75ms.

- Le taux de perte des paquets : La perte d'un paquet entraîne la disparition d'un ou plusieurs échantillons du flux de voix, on parle alors de *distorsions du signal*. Suivant le nombre de paquets perdus, la qualité sonore en bout de ligne peut s'en ressentir. Des solutions de retransmission des paquets de voix engendreraient des délais trop importants. Si les paquets perdus ne le sont pas en rafales, les codecs sont capables de « reconstruire » les échantillons manquants, via des algorithmes prédictifs, du moins jusqu'à un certain seuil. Ce seuil est traduit en taux de pertes. La valeur optimale du taux de perte, couramment admise pour un service de VoIP, est inférieure ou égale à 1% et la valeur acceptable est inférieure ou égale à 3%.

Ainsi le délai total se résume à l'équation suivante :

Délai total = délai de propagation + délai de transport + délai de paquétisation + délai de la gigue.

En cas de trafic crypté, le délai relatif à ce trafic augmente plus rapidement que le délai d'un trafic en clair acheminé dans les mêmes conditions. Ceci est causé par le temps introduit par le cryptage. L'impact des différents algorithmes de cryptage devient négligeable lorsque la taille des paquets augmente. Mais le délai de transmission, les délais internes aux routeurs (calcul du checksum), les délais de mémorisation augmentent proportionnellement avec l'augmentation de la taille des paquets. Ce qui affectent aussi la variation de délai (gigue) et le délai total délai livraison du paquet au destinataire.

La sécurité est donc aussi bien nécessaire aux informations de voix qu'aux informations de signalisation pour l'établissement de l'appel et aux services, qui transportent des informations critiques comme l'identifiant du numéro appelé et appelant, etc. Or, introduire une nouvelle couche de sécurité, garantissant la confidentialité, l'intégrité et l'authentification, ralentit la transmission des paquets et augmente le délai comme nous le verrons dans les solutions de sécurité disponible dans le réseau IP.

Actuellement, les concepteurs des applications de voix sur IP sont confrontés à un nombre de choix non satisfaisant pour fournir la sécurité. D'abord, ils peuvent employer IPSec. Cependant, IPSec n'est pas bien adapté aux modèles d'application client-serveur et est difficile à implémenter avec des applications puisqu'il fonctionne dans le noyau de la machine (le *kernel*). D'un point de vue pratique, IPSec est un protocole relativement difficile à implémenter d'une part à cause de sa complexité intrinsèque (multiples sous-protocoles...) et d'autre part à cause de ses interactions avec les processus réseaux courants. Ce qui rend ce standard assez lourd et compliqué à implémenter et à maintenir dans un environnement de la téléphonie sur IP. En effet, un paquet IP régulier transportant de la voix a une taille de 60 octets (entête IP = 20 octets, entête UDP = 8 octets, entête RTP = 12 octets, payload = 20 octets). L'utilisation de IPsec augmente la taille des paquets pour atteindre 100 à 110 octets selon l'algorithme de cryptage utilisé. En conséquence, le rapport donnée utile actuelle vs taille totale des paquets diminue, indiquant une perte de la bande passante utile [VoIPsec]. Les aspects négatifs qui affectent son utilisation pour la voix sont en relation avec les besoins en ressource; la charge du processeur augmente énormément due au chiffrement, l'*overhead* induit et donc le débit effectif... ainsi il faut réfléchir à diminuer la taille des paquets en appliquant une compression des entêtes des paquets IPsec.

Ces mêmes concepteurs peuvent concevoir un protocole de sécurité fait spécialement pour la couche application. Le protocole SIP, par exemple, emploie une variante de S/MIME [RFC2633] pour sécuriser le trafic. L'implémentation de S/MIME dans SIP a pris énormément d'effort. Une autre alternative serait de régénérer l'application sur TCP et d'utiliser TLS. Malheureusement beaucoup de telles applications dépendent de la sémantique de datagramme et ont une performance très médiocre et inacceptable quand ils fonctionnent au dessus de TCP. La performance de TLS dépend de plusieurs facteurs :

- L'algorithme d'échange de clef, RSA ou DH.
- La taille de la clé de chiffrement/déchiffrement.
- L'algorithme de chiffrement symétrique, RC4, 3DES, AES, etc.
- La taille du *Record*.
- La capacité du microprocesseur client/serveur, le débit du réseau et la vitesse du traitement des données *Handshake.*

Comme nous l'avons déjà cité, TLS utilise une combinaison d'algorithmes à clé publique, d'algorithme de chiffrement symétrique et d'une fonction de hachage. La charge d'une session TLS dépend de l'algorithme de chiffrement choisi par le client et le serveur, de la taille des clés et la chaîne de certification, et du temps de traitement des messages protocolaires TLS [Bad04].

DTLS introduit environ 35% *d'overhead* [Mod04] durant la phase de négociation (ou *handshake*) du protocole. Cet *overhead* des données transportées dans les messages (ou *records*) est réduit à cause de l'absence des entêtes de fragmentation.

La charge cryptographique et protocolaire du TLS/DTLS peut se réduire en introduisant d'autres mécanismes d'authentification que les certificats à clé publique et en définissant d'autres architectures. Notons ici que le stockage des clés secrètes, des clés privées et des certificats des tiers de confiance (CA) reste une mission importante pour les systèmes des clients et des serveurs. Cependant, l'utilisation de la carte à puce peut réduire au minimum cette problématique.

Due aux contraintes temporelles citées ci haut, la taille des paquets de voix RTP se limitera entre 10 et 50 bytes de données [Bar02] utiles afin de garantir une bonne performance et respecter les contraintes temporelles. La durée du *payload* est un compromis entre le besoin en bande passante et la qualité de la voix. SRTP ajoute 4 à 10 bytes à la taille du paquet RTP pour crypter les données utiles provenant du codec. En prenant un échantillon de voix sur 20ms, la taille du paquet RTP se réduira alors à 20 octets. Si nous faisons des calculs théoriques avec SRTP, nous obtiendrons les résultats suivants :

Table 1- Ratio du *payload* sur la taille totale du paquet

Type de Paquet	Entête (octet)	Taille du paquet (octet)	Ratio
IP	40	60	34 %
IP + SRTP (4 bytes)	44	64	31 %

On remarque que *l'overhead* introduit par l'utilisation de SRTP n'affecte pas trop le rapport entête/taille du paquet SRTP. Une simulation sur NS de SRTP sécurisant un appel de voix effectué avec H.323 montre que *l'overhead* engendré par les champs de SRTP ont un effet négligeable lorsque la taille du paquet RTP augmente. Un délai est constaté dû au fait des opérations de modification de l'entête RTP pour inclure les champs spécifiques à SRTP.
On constate du graphe ci-dessous que plus la taille du paquet sur la ligne est petite plus le nombre de paquets envoyés sur la ligne diffère dans les deux cas (avec et sans SRTP). Plus la taille du paquet est grand l'effet de SRTP devient négligeable.
Cela est tout a fait logique, puisque les 8 octets ajoutées à chaque paquet seront un énorme overhead pour les paquets de petite taille et seront négligeables pour les paquets de grande taille.

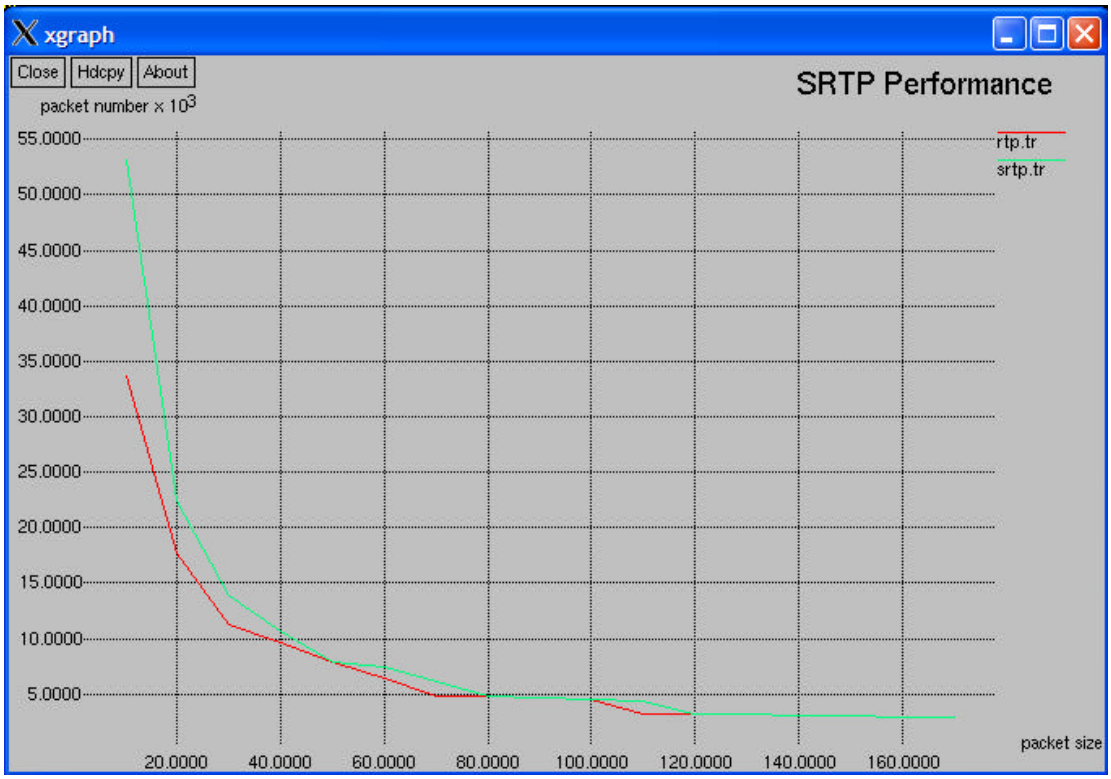

Figure 2.14 – graphe simulant l'overhead de SRTP avec l'outil *Network Simulator (NS)*

Toutefois, SRTP laisse la possibilité d'utiliser un algorithme de compression de l'entête [RFC3545] IP/UDP/SRTP pour optimiser l'utilisation de la bande passante disponible à la VoIP tout en sécurisant les informations échangées. Ces propriétés assurent que SRTP est un bon schéma de sécurité pour le trafic RTP quel que soit l'environnement fixe ou sans fils. Cependant SRTP présente quelques points faibles :

- SRTP n'adresse aucune sécurité de la signalisation. Ce qui requiert un mécanisme séparé pour tous les autres types de communications ;

- besoin d'une gestion de clé séparée, tel IKE, ISAKMP/Oakley, Kerberos, Diffie-Hellman ou MIKEY [RFC3830]; des mesures [Bil03] des délais induits par la génération des clés utilisant Diffie-Hellman et MIKEY ont montré une augmentation assez importante des délais lors de l'utilisation de Diffie-Hellman par rapport aux délais introduits par MIKEY;
- Nécessité d'introduire des modifications dans les implémentations existantes de la voix sur IP pour supporter SRTP;
- manque d'authentification des utilisateurs dans des sessions RTP groupées ou multicast.

2.7.1 Solution de sécurité de bout en bout : FNBDT

2.7.1.1 Introduction

En 1999, la National Security Agency américaine a réalisé une interopérabilité sécurisée entre les systèmes à fil et sans fil lorsqu'elle a créé un consortium industriel et gouvernemental qui a convenu d'un protocole commun de signalisation, appelé FNBDT [FNBDT] (*Future Narrow Band Digital Terminal*), qui a été également développé pour fournir un schéma de signalisation permettant aux utilisateurs de communiquer en toute sécurité avec d'autres produits compatibles.

Au lieu de développer différents produits individuels de communications sécurisés, conçus pour interopérer entre eux, FNBDT définit une architecture interopérable et sécurisée indépendante de la technologie du réseau sous-jacent. Ainsi FNBDT est basé sur des normes interopérables permettant des communications de données et de voix sécurisées de bout en bout à travers une infrastructure hétérogène de réseaux (PSTN, RNIS et cellulaires, etc.).

FNBDT inclut des possibilités communes de traitement de la voix, un protocole de signalisation commun, une base commune d'algorithme cryptographique, et un processus commun de gestion de clés, chacun défini par un document différent :

- FNBDT-120 : Plan de Gestion des Clés ;
- FNBDT-210 : Plan de Signalisation (Version Formelle 1.1)
- FNBDT-230 : Caractéristiques Cryptographiques (sous forme d'ébauche – *draft*)
- FNBDT-220 : Conditions d'Interopérabilité (non officiel – *Informal*)
- FNBDT-6xx : Documentation d'Essai (discussion en suspens)

Dans ce qui suit, nous étudierons le protocole de signalisation de FNBDT comme introduit dans le document FNBDT-210 version 1.1.

2.7.1.2 Les services de sécurité de FNBDT

La signalisation de FNBDT est initialement définie pour les terminaux qui opèrent au-dessus de canaux à bande numérique étroite tels que les cellulaires numériques commerciaux (CDMA de premier intérêt, GSM, TDMA, etc.), les satellites mobile (IRIDIUMTM, Globalstar, ICO) et les canaux militaires et tactiques. Mais en dépit de son nom, la signalisation FNBDT est également compatible avec les canaux à large bande tels que l'Internet et l'ATM.

Le plan de signalisation de FNBDT utilise pour la définition de sa norme, les couches 5 (Session) et 6 (Présentation) de l'OSI (Open Systems Interconnexion) (cf. fig.

2.15). La couche Application contient des fonctions pour des services particuliers d'applications, tels que le transfert de fichier et l'accès à distance. Une application FNBDT utilise la technique de chiffrement Rijndael (connue aussi sous le nom de Advanced Encryption Standard (AES)) en mode compteur (Counter-Mode)). FNBDT compte sur les développeurs d'applications pour spécifier les couches inférieures du réseau (couches OSI 1 à 4) afin de fournir la livraison de bout en bout des données.

En ne délimitant pas les couches du réseau, FNBDT fournit la plus grande flexibilité pour l'inclusion d'une variété de réseaux différents, et même des concaténations de réseaux différents. FNBDT utilise la signalisation de la couche application insérée dans une couche de transport fiable de données. Des fonctions d'interactions fonctions (IWF pour *Interworking Functions*) permettent l'interfaçage entre des réseaux hétérogènes.

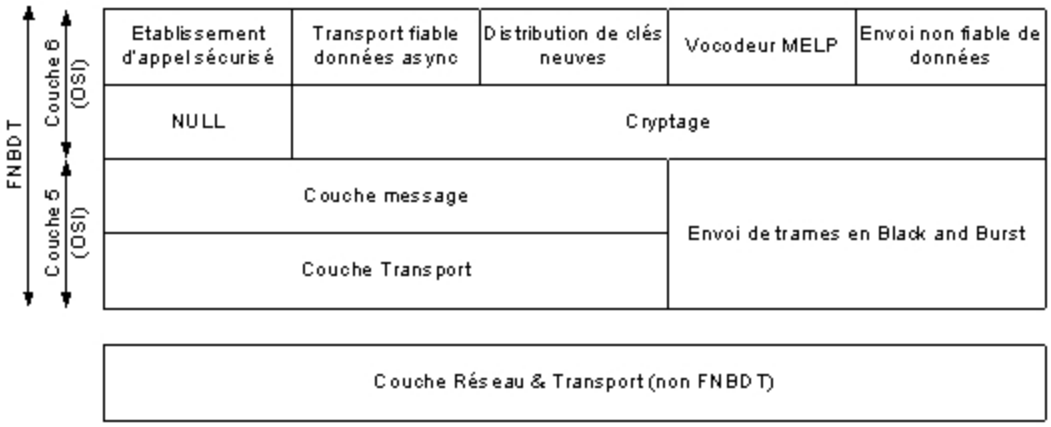

Figure 2.15 - Les couches protocolaires de FNBDT.

La figure 2.15 montre l'architecture de FNBDT. Les différentes couches et modes de fonctionnement sont définis, du bas vers le haut, comme suit :

- les couches physiques, réseau et transport sous-jacents sont indépendants de FNBDT ;
- la couche *Transport* fournit un service de transport fiable pour les données avec accusé de réception. En d'autres termes, toute donnée perdue pendant la transmission devrait être renvoyée ;
- la couche *Message* permet de reconstruire les commandes issues de la couche application en des messages avant leur transmission à la couche en dessous. Elle s'occupe aussi de l'établissement et du contrôle d'appel ;
- la couche Blank & Burst permet l'envoi de trames de voix et de données sécurisées avec une tolérance aux erreurs, des rejets et des évanouissements. Les données cryptographiques de synchronisation doivent être disponibles dans cette couche de façon fiable. Il n'y a pas d'espace pour la mise en trame et les retransmissions dans les données de voix. Le concept du système ne permet pas la mise en trame et les retransmissions des données cryptographiques de synchronisation. L'approche de gestion de synchronisation réalise la

récupération de l'information de synchronisation en utilisant un vecteur d'état non classifié ;

- la couche *Cryptage* chiffre et déchiffre les données échangées ;
- le niveau le plus haut est composé de différents blocs d'application tels que l'établissement d'appel sécurisé, le transport fiable de données asynchrones, le renouvellement électronique des clés, le vocodeur MELP à 2400bps et les données asynchrones à débit garanti (c'est l'envoi non fiable de données).

La signalisation de FNBDT peut être transportée d'une manière transparente au-dessus d'un réseau de téléphonie. Les services de sécurité offerts par FNBDT sont :

- Gestion des clés : FNBDT offre d'utiliser le système FIREFLY développé par la NSA.
- L'authentification : Avec FNBDT, l'authentification des utilisateurs est implicite à travers l'ensemble des messages échangés lors de l'établissement d'un appel sécurisé : échange des clés entre les deux utilisateurs, suivi d'une phase de spécification des paramètres cryptographiques et d'une phase de test de ces paramètres. Ce test permet de s'assurer que les deux utilisateurs utilisent bien l'algorithme de cryptage et les clés négociés entre les points d'extrémité.
- La confidentialité : Dans FNBDT, la sécurité est adressée par l'adoption du modèle de PKI/KMI pour l'échange des clés cryptographiques. Un document spécifique est dédié à la définition de la couche cryptographique utilisée dans FNBDT, mais n'est pas accessible au public.
- Intégrité: L'intégrité assurée avec FNBDT n'est pas cryptographique puisqu'elle se base uniquement sur le contrôle du CRC apposé à certains messages et non pas sur le résultat d'une fonction cryptographique.
- Non rejeu: Dans FNBDT, chaque message transporte une identification unique de la source (*Source ID*) des messages définis. Cette identification pourrait être utilisée pour établir une liste de rejeu soumise à un mécanisme de fenêtre coulissante, à vérifier par le récepteur afin de se protéger contre les attaques par rejeu.

2.7.1.3 Caractéristiques de FNBDT

Nous pouvons dégager les caractéristiques suivants et qui sont propres au protocole FNBDT :

- FNBDT introduit une signalisation propre à lui qui renforce son indépendance par rapport aux couches de réseau sous-jacentes. Il permet ainsi une interopérabilité sécurisée entre équipements de tout genre et entre réseaux hétérogènes;
- FNBDT permet de sécuriser de bout en bout les communications de voix et de données, tout en offrant aussi la possibilité de transmettre la voix en clair ;
- FNBDT prévoit un arrêt en douce des communications en cours sans éprouver le besoin de les avoir terminées. En effet, des trames spécifiques permettent une réinitialisation et une re-synchronisation du système en cas de problèmes.
- FNBDT présente une grande tolérance aux pertes de trames grâce aux acquittements positifs et réémissions sélectives;

- FNBDT présente une grande tolérance aux déséquencements puisque côté récepteur, des mémoires permettent le stockage des trames déséquencées pour une livraison correcte,
- FNBDT est à la base défini pour être déployé sur tout type de réseaux, il apporte avec lui ses propres mécanismes de détection d'erreurs, de retransmission, et de contrôle pour fiabiliser la transmission et acquitter la bonne réception des messages.

2.7.1.4 Les inconvénients

FNBDT ne garantit malheureusement pas tous les mécanismes de sécurité dont les documents de spécifications sont classifiés par la NSA. Les inconvénients de FNBDT sont montrés par les points suivants :

- Protocole fermé gouverné par la NSA et les pays du *NATO*.
- Possibilité d'attaque par homme du milieu si les certificats sont envoyés en clair entre user A et user B.
- FNBDT présente une surcharge protocolaire (*overhead*) due à la taille fixe des trames. En effet, un message en mode tramé est délimité par 2 fanions de 8 octets chacun et peut être formé de 1 à 255 trames, chacune de 20 octets dont seuls 13 d'utiles, ce qui fait une surcharge protocolaire :
  - dans le cas d'une trame : on envoie 13 octets utiles sur (2*8 + 20), donc seuls 36% des octets envoyés sont utiles, ce qui fait une surcharge protocolaire de près de 64% ;
  - dans le cas de 255 trames : on envoie 255*13 octets utiles sur (2*8 + 255*20), donc 64,8% des octets envoyés sont utiles, ce qui réduit la charge protocolaire à près de 35% ;
  - il est à noter que cet *overhead* affecte uniquement les messages de signalisation. Une fois les paramètres du protocole ont été négociés, le mode tramé bascule en mode non tramé à plein débit pour transporter la voix.

2.7.1.5 Conclusion

Tracé l'évolution des la sécurisation des communications depuis le système SIGSALY puis STU-II jusqu'à FNBDT, nous voyons bien la nécessité d'offrir des communications interopérables et sécurisées. Il est évident que la solution FNBDT offre de nouvelles opportunités dans ce domaine surtout que des produits commencent à être disponible sur le marché. FNBDT représente un paradigme fondamental dans la désignation d'une solution générique pouvant être utilisé au niveau applicatif des couches du modèle de l'OSI et donc faisant abstraction du réseau de transport.

2.8 Conclusion générale du chapitre

La table 2 récapitule la majorité des infrastructures de téléphonie ainsi que les solutions et les mécanismes de sécurité utilisés par chaque solution. Nous remarquons que la majorité des solutions utilisent les certificats pour assurer l'authentification de l'usager lui-même et non pas uniquement l'authentification de l'équipement. Cependant les certificats signés par les usagers eux-mêmes (ou *self-signed*) ne sont pas sécurisés

contre les attaques de l'homme du milieu (MitM). Pour cela, il vaut mieux utiliser une autorité de certification publique et bien connue. Il faut ensuite que chaque usager s'assure de la validité du certificat reçu auprès de cette autorité.

Dans ce chapitre, nous avons parcouru succinctement les solutions de sécurité inhérentes à chaque type de réseau offrant un service de voix. Nous avons montré que les réseaux téléphoniques traditionnels n'offrent aucune sécurité à ses abonnés, sauf des appareils téléphoniques spécifiques qui cryptent le trafic de voix. L'architecture NATO NBSV-II opère sur une infrastructure de transport homogène mais n'assure pas la sécurisation dur réseau numérique bout en bout Actuellement, les développements dans en technologies de communications (CDMA, TDMA, IP, ATM, etc.) ont fragmenté cette infrastructure résultant en des solutions de sécurité non interopérables. Ainsi, Les réseaux mobiles centrent leur intérêt pour la sécurité essentiellement sur le lien radio en authentifiant les identités des abonnés et en chiffrant le trafic traversant ce lien. Pour la voix sur IP, les deux standards protagonistes sont H.323 de l'UIT-T et SIP de l'IETF. Chacun propose des solutions de sécurité propre à lui ou bien utilise les solutions de sécurité offerte par les couches transport (TLS, DTLS, SRTP) et réseau (IPsec) du réseau IP. Donc, nous voyons bien que chaque réseau utilise ses propres moyens pour sécuriser le trafic de voix, certains services de sécurité sont offerts alors que d'autres n'existent pas. Pour palier à cette lacune dans la possibilité de sécuriser le trafic de téléphonie de bout en bout et dans un monde d'interconnexion et de croisement de différents types de réseaux, l'agence américaine de sécurité (NSA) offre une solution de sécurité, de bout en bout, indépendante des couches transport et réseau et transparente par rapport à l'utilisateur. Dans le chapitre suivant, nous proposerons une architecture et les services de sécurité appropriés pour la téléphonie.

Table 2 - Solutions et mécanismes de sécurité

Sécurité (protocoles et mécanismes)	Authentification	Confidentialité	Intégrité	Non rejeu	Non-Répudiation	Observations
HTTP 1.1 Digest Authentification	PSK	-	-	-	-	Utilisé par SIP
S/MIME	PKI	✓	✓			Utilisé par SIP
TLS	PKI	✓	✓	✓	✓	Utilisé par TCP
DTLS	PKI	✓	✓	✓	✓	Utilisé par UDP
IPsec	PKI	✓	✓	✓	-	Utilisé avec IP
SRTP	PSK	✓	✓	✓	-	Utilisé par SRTP
H.235	PKI PSK	✓	✓	-	✓	Utilisé par H.323
Skype	PKI	✓	-	-	-	Application P2P
FNBDT	PKI	✓	-	✓	-	Application de bout en bout
GSM	PSK	✓	-	-	-	
GPRS	PSK	✓	-	-	-	
UMTS	PSK	✓	✓	-	-	
Bluetooth	PSK	✓	-	-	-	
WLAN	PKI PSK	✓	✓*	-	-	

***** ces services de sécurité dépendent de la solution utilisée avec WLAN

Chapitre 3

3 Architecture pour la sécurisation de la téléphonie

3.1 Résumé

Pour assurer une solution de sécurité de bout en bout pour la téléphonie, il est nécessaire de définir une architecture de sécurité qui prend en compte les spécificités des réseaux de téléphonie. Le but de ce chapitre est donc de tirer une architecture abstraite de la téléphonie basée sur les différentes architectures des réseaux de téléphonie actuels. Nous dégagerons ensuite une architecture de sécurité avec les services nécessaire à la sécurisation de l'appel téléphonique. Pour rendre cette solution de sécurité transparente et indépendante du réseau sous-jacent, une interface (API) est insérée entre la solution de sécurité et le réseau sous-jacent.

3.2 Motivations

La sécurisation de la téléphonie commence à prendre sa lancée dans les années quarante lors de la deuxième guerre mondiale. A cette époque, la Bell *Telephone Laboratories* commence les travaux sur le premier système téléphonique sécurisé [Sigsaly].

La déréglementation et la mondialisation des télécommunications ont ouvert la porte à l'émergence de nouveaux réseaux de téléphonie et à leur diversification. Nombreuses sont les solutions de sécurité offertes par ses réseaux qui permettent de se prémunir plus ou moins contre d'éventuelles attaques. Ces solutions sont toutefois propriétaires de chaque type de réseau, et jusque là, d'aucune n'est polyvalente, notamment en matière de sécurité. Les recherches dans le domaine de la sécurité de la téléphonie se trouvent de facto orientées vers le développement d'une solution indépendante de l'infrastructure (fixe ou sans fil) et indépendante de la sécurisation offerte par les couches sous-jacentes. Ainsi, les efforts se sont poursuivis par l'agence de sécurité américaine, la NSA, pour offrir le protocole FNBDT. Cependant, FNBDT est une solution fermée dont les spécifications ne sont pas diffusées au grand public.

Une architecture de sécurité doit donc tendre à être ouverte, interopérable, englobant la majorité des services de sécurité requis, transparente à l'utilisateur et accessible à tous les particuliers. L'objectif est donc de découpler la fourniture des services de sécurité pour la téléphonie du réseau de transport de manière indépendante du mode d'accès. Il s'agit alors de disposer d'une architecture de création, d'intégration et d'exécution de services de sécurité qui constituera un standard pour les plates-formes ouvertes et interopérables de la téléphonie fixe et mobile. Cette architecture de sécurité sépare les services de sécurité de l'infrastructure matérielle et logicielle du réseau sous-jacent de l'opérateur en se plaçant au niveau de la couche applicative et en articulant son

interfonctionnement avec son environnement téléphonique autour d'interfaces ouvertes. La création de services de sécurité modulaires et portables est ainsi grandement facilitée.

3.3 Architecture abstraite du service de téléphonie

Le réseau téléphonique constitue un des plus grands réseaux au monde. Essentiellement analogique au départ (RTC), le réseau s'est progressivement numérisé (RNIS) : la transmission dans le réseau d'abord, suivie par la commutation ensuite. Le besoin de communiquer avec une personne en déplacement a conduit aux concepts de radiotéléphonie cellulaire (GSM). La mutation en cours vers le multimédia fixe (VoIP) et mobile (UMTS) se caractérise par la nécessité de gérer la disponibilité des ressources, l'adaptation à la demande et au profil de l'utilisateur, la mobilité, ainsi que l'assurance de l'identité du correspondant et de la confidentialité des échanges.

3.3.1 Architecture actuelle des réseaux de téléphonie

Relier chacun des abonnés à tous les autres nécessiterait un écheveau de milliards de liaisons, ce qui est irréalisable. Aussi des aiguillages entre les différents niveaux de jonction du réseau ont été prévus. Les réseaux à commutation de circuit tel le réseau RTC et le réseau RNIS [Rig98] se basent sur une architecture comprenant des commutateurs et des liaisons de transmission reliant d'un côté ces commutateurs entre eux et d'un autre côté ces commutateurs aux terminaux téléphoniques. Le cœur d'un réseau téléphonique comprend des commutateurs locaux sur lesquels sont connectés les terminaux usagers et des commutateurs de transit véhiculant le trafic entre commutateurs locaux.

Nous retrouvons une architecture similaire dans les réseaux mobiles à une différence près que les liaisons entre l'usager mobile et le point d'accès au réseau et donc aux commutateurs du cœur du réseau sont devenues des interfaces radio qui nécessiteraient un matériel spécifique pour la transmission aérienne et pour la gestion de la mobilité de l'usager.

Ces réseaux peuvent également inclure des équipements de fourniture et de gestion de services tels que les serveurs de messagerie (vocale et texte) et un réseau intelligent (IN, *intelligent network*) pour l'offerte de nouveaux services à valeur ajoutée.

En offrant la possibilité de faire transiter les communications voix sur des réseaux de données, la ToIP marque la fin d'une cohabitation entre deux réseaux distincts, très différents, du point de vue de la gestion et de la technologie. Plusieurs standards existent à l'heure actuelle et qui offre les services de la voix sur IP. Nous citerons le standard H.323 de l'UIT-T et le standard SIP de l'IETF. Ces standards assureront la signalisation des appels qui nécessiterait l'utilisation des outils matériels et logiciels qui vont gérer l'accès des usagers et les paramètres de l'appel comme le Gatekeeper de H.323 ou le Registrar de SIP. D'autres protocoles (RTP/RTCP) seront utilisés pour transporter les flux de parole.

3.3.2 Séparation Plan usager/Plan contrôle

Un appel comporte une conversation de signalisation au préalable nécessaire à l'établissement de l'appel, à la réservation des ressources, et à leur relâchement, puis la conversation des usagers. Les spécialistes de la commutation ont pris l'habitude de distinguer ces deux types d'appels qui constituent l'appel global. Ils représentent dans un

plan la fonction de traitement d'appel. Ce plan est appelé le « plan de contrôle ». Ils représentent dans un autre plan les usagers et les ressources qui les relient (lignes, réseaux de connexion, jonctions). Ce deuxième plan est appelé le «plan usager ». On peut dire que les usagers eux-mêmes sont partagés entre chacun des plans. Ils communiquent, en effet, d'abord avec le traitement d'appel et se trouvent dans le plan contrôle, puis avec l'autre usager et se trouvent dans le plan usager.
Ainsi la ligne de téléphone porte des informations de deux plans :

- La signalisation (conversation avec la fonction de traitement d'appel) qui appartient au plan contrôle
- La conversation entre l'usager demandeur et l'usager demandé qui appartient au plan usager

La représentation des deux composantes de l'appel en deux plans différents devient encore plus claire si nous considérons un réseau de commutateurs, chaque commutateur étant évidemment doté de sa fonction de traitement d'appel. Cependant, les commutateurs utilisés assurent les deux fonctions. Ainsi, en plus de la fonction de connecter les circuits de voix à d'autres commutateurs, ils doivent aussi assurer la fonction de signalisation. Afin d'alléger le travail des commutateurs et améliorer leurs performances, il est nécessaire de séparer le plan contrôle du plan usager, notion qui a été introduite par le réseau de signalisation dans les réseaux de téléphonie fixe et mobile. Dans les réseaux voip, cette notion de séparation des flux de signalisation et de données est maintenue par l'utilisation de protocoles spécifiques pour transporter les messages de signalisation (H.323, SIP) et d'autres protocoles pour le transport des conversations des usagers (RTP/RTCP).

3.3.3 Architecture abstraite de la téléphonie

De cette description succincte des architectures de téléphonie, nous tirerons une architecture abstraite qui consiste en l'utilisation de terminaux permettant aux usagers de converser entre eux. Ces terminaux masquent le cœur du réseau qui est constitué des nœuds de transfert pour assurer la signalisation d'une part et la commutation/routage des appels d'autre part. Des serveurs peuvent exister dans le cœur du réseau de téléphonie pour assurer de nouveaux services aux utilisateurs dont le service de sécurité. Ainsi, nous pouvons séparer dans cette architecture la couche réseau qui comprendra les réseaux PSTN, les PLMN, et les réseaux en mode paquet de la couche service qui offrira le service de sécurité aux différentes plates-formes de téléphonie. Au dessus de la couche de service se place la couche applicative de sécurité qui est indépendante des protocoles sous-jacents. Une couche protocolaire se positionne entre la couche service et la couche réseau pour articuler les différents réseaux de téléphonie.

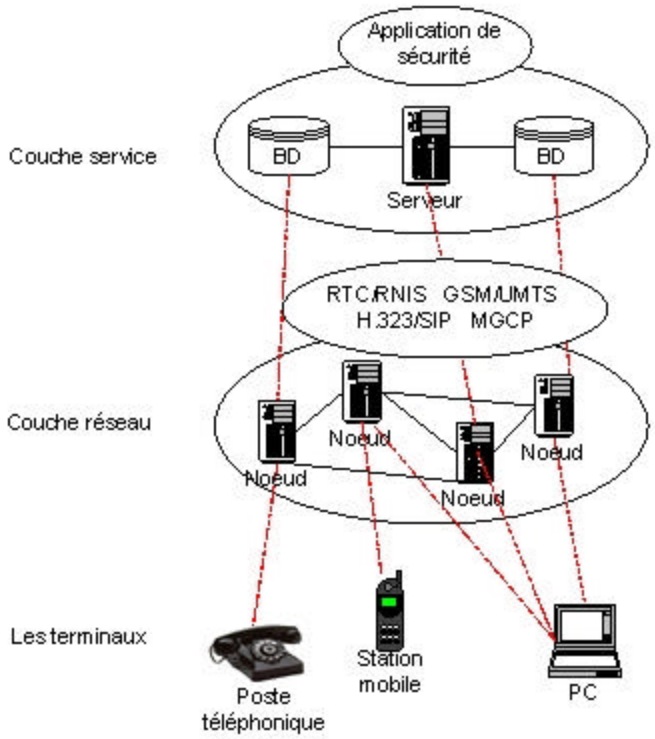

Figure 3.1 - synoptique d'un réseau de téléphonie

3.4 Les services de sécurité pour la téléphonie

Une classification utile des services de sécurité nécessaire à la téléphonie est la suivante :

- Authentification de l'usager;
- Intégrité ;
- confidentialité ;
- non-rejeu ;
- non-répudiation de l'appel.

3.4.1 Authentification de l'usager

L'authentification consiste à s'assurer de l'identité de la personne avec qui nous communiquons. Dans le cas des applications informatiques, la fonction du service d'authentification est d'assurer le destinataire que le message a bien pour origine la source dont il prétend être issu. Dans le cas d'une conversation téléphonique, le service assure lors de l'initialisation de la connexion que les deux correspondants sont authentiques. Deux aspects du service d'authentification peuvent se présenter :

- Une authentification locale qui se déroule entre l'usager et une entité tierce locale.
- Une authentification globale qui peut se subdiviser en :
  - Une authentification entre deux usagers de bout en bout.
  - Une authentification entre deux usagers à travers une entité tierce de confiance commune

Ensuite, le service doit assurer que cette connexion n'est pas perturbée par une tierce partie qui pourrait se faire passer pour une des deux entités légitimes à des fins de transmission, de réception ou de détournement d'information non autorisées.

3.4.2 Intégrité

Un service d'intégrité traitant les flots de messages de paroles ou de signalisation d'un appel téléphonique assure que ces messages soient reçus par le destinataire sans modification ou altération. La destruction des données est également traitée par ce service. Ce service se rapporte sur la détection plutôt que sur la prévention contre les attaques à l'intégrité des informations acheminées. Si une violation d'intégrité est détectée, alors le service peut simplement rapporter cette violation, et d'autres logiciels ou une intervention humaine sera requise pour la reprise sur violation. L'incorporation de mécanismes automatiques de reprise, est, en général, la meilleure solution.

Deux mécanismes existent pour assurer l'intégrité des données transportées : soit une fonction de hachage unidirectionnelle ou bien des signatures numériques.

3.4.3 Confidentialité

Le service de confidentialité consiste à masquer les informations à des tierces parties en les chiffrant, et en rendant ces informations compréhensibles uniquement au destinataire authentique qui possède la bonne clé pour les déchiffrer. Selon les besoins, plusieurs niveaux de protection sont identifiables. Le service le plus général est de chiffrer toutes les informations que ce soit les données de signalisation ou de parole. Des formes restreintes de ce service peuvent également être définies, incluant uniquement la protection des données acheminant les informations nécessaires à l'authentification des usagers, pour notamment assurer la protection de l'identité des communicants. L'usager a le choix de chiffrer les données de parole ou de les transporter en clair. Ses raffinements nécessitent la gestion de politique de sécurité relative aux besoins de sécurité des usagers et au contrat de services signé avec le fournisseur de service de sécurité.

Un autre aspect de la confidentialité est la protection du flot de données contre l'analyse. Cela requiert qu'un attaquant ne puisse observer les sources et destinations, les fréquences, longueurs ou autres caractéristiques du trafic existant sur un équipement de communication.

Deux approches principales existent pour assurer la confidentialité : cryptographie symétrique ou asymétrique.

3.4.4 Non rejeu

Le rejeu implique la capture passive des données de signalisation ou de parole relatif à l'appel et leur retransmission ultérieure en vue de produire un effet non autorisé. Pour assurer la protection contre les attaques par rejeu, on peut associer un *timestamp* aux messages transportant des données non modifiables par des tierces parties. Dans un contexte de transfert de données en mode datagramme, on utilise en général une fenêtre glissante permettant de préciser les messages déjà reçus et authentifiés.

3.4.5 Non répudiation de l'appel

La non-répudiation empêche tant l'appelant que l'appelé de nier avoir participé à l'appel. Ainsi, lorsqu'un appel téléphonique est établi entre deux ou plusieurs usagers, les participants dans cet appel devraient être capable de prouver que son ou ses correspondants ont bien participé à la communication.

En téléphonie, pour s'assurer que l'usager a bien participé à un appel avec tel correspondant à telle date, nous avons recours aux factures émises par l'opérateur. Cependant, ces factures font preuve de l'identifiant de la ligne de communication de laquelle l'appel a été établi. Pour rendre cette facture fiable et authentique et inclure l'identité des usagers, il suffit que ces rapports soient émis par l'usager lui-même ou bien par une entité tierce de confiance, signés par l'entité émettrice du dudit rapport.

3.5 Typologie des solutions, exigences et contraintes d'une sécurité de la téléphonie globale

Comme on vient de le voir dans le paragraphe précèdent, les services de sécurité attendus sont ceux déjà usuels dans le contexte des applications classiques. Dans le contexte de la téléphonie, nous sommes confrontés à différentes variantes de cette application de téléphonie. Chacune de ces variantes apporte sa propre manière de réaliser ces services de sécurité. Nous avons analysé ces solutions précédemment. Il s'agit pour nous d'atteindre une solution assurant l'ensemble de ces services indépendamment de l'infrastructure de téléphonie sous jacente.

Quatre solutions de sécurité pour la téléphonie peuvent être envisagés :

- Solution transparente. La nature de la pile de communication est telle qu'un protocole de niveau N ne fait aucune hypothèse sur le protocole de niveau N+1. On peut tout à fait insérer entre deux couches un protocole assurant des services de sécurité sans aucune modification des machines protocolaires adjacentes. Il est néanmoins nécessaire de revoir les interfaces.
- Solution par signalisation. Cette solution comme la précédente ne remet pas en cause l'existant et donc n'implique aucune modification sur les protocoles en cours. Il s'agit d'associer à un niveau donné de la pile de communication un ou plusieurs protocoles de signalisation qui couvre les services de sécurité de la téléphonie. Ces protocoles peuvent soit empruntés un réseau distinct soit le même réseau voir même être encapsulés par les protocoles de téléphonie sous jacent.
- Solution par extension. Ces solutions consistent à étendre les protocoles existants avec les services de sécurité manquants. Ces extensions doivent être supportées par les différentes infrastructures de téléphonie. Par exemple il s'agira de modifier la signalisation associée à chacune des infrastructures. Cette solution induit des modifications au travers de toutes les infrastructures traversées.
- Solution par traitement locale. Cette solution ne peut pas assurer les services de sécurité cités plus haut. Car dans ce contexte la communication est l'objectif de base de l'application. Dans un contexte d'application classique ce type d'approche consiste notamment à assurer un service de filtrage ou de contrôle de conformité d'un trafic.

Si nous analysons chacune des solutions citées précédemment, nous nous retrouvons devant une sécurité locale effectuée au niveau de chaque réseau de téléphonie n'engendrant pas une sécurisation de bout en bout qui est le but final de nos travaux.
Une modification des protocoles existants peut freiner le déploiement et causer une non interopérabilité. On ne peut pas ainsi augmenter les protocoles existants par de nouveaux services de sécurité.
Un nouveau protocole de signalisation couvrant les services de sécurité nécessiterait le passage par un réseau distinct et donc il faut non seulement assurer la sécurisation du flux téléphonique mais aussi bien s'assurer de la fiabilité et de la sécurité du réseau de transport. Par contre, l'encapsulation de ce protocole de signalisation dans les protocoles de téléphonie sous jacent conduirait à une duplication des services de sécurité offerts par ce protocole et les protocoles de sécurité qui accompagnent le réseau de téléphonie notamment dans le réseau IP.

L'insertion d'une couche de sécurité intermédiaire induirait en une abstraction des infrastructures de téléphonie sous jacente et ne nécessiterait aucune modification des protocoles des couches inférieures. Toutefois, deux couches adjacentes communiquent à travers d'interfaces de service (SDU, pour *Service Data Unit*). Il est nécessaire alors de revoir ses interfaces ou de définir une nouvelle API qui s'insèrera entre les deux couches N et N+1. Cette API permet à la couche applicative de sécurité de s'interfacer avec les couches protocolaires du réseau de téléphonie sous-jacent mais aussi d'interopérer avec les solutions de sécurité existantes dans ce réseau afin d'éviter la duplication des services. Cette interopérabilité n'entre pas en conflit avec la transparence de la solution de sécurité mais empêchera la redondance des services de sécurité. Il est à noter qu'introduire de la sécurité augmentera les délais et par la suite affectera la qualité de la communication. Il est nécessaire alors d'établir un compromis entre le degré de sécurité requis et les contraintes de la téléphonie en terme de qualité de service et de délais.

Finalement, afin d'assurer la sécurisation de la téléphonie, il est nécessaire de revoir les phases d'un appel téléphonique. Comme nous l'avons vu dans le paragraphe précédent sur l'architecture du réseau téléphonique, cette dernière sépare l'appel en deux plans distincts : le plan de signalisation qui transporte les messages de commandes et de contrôle requis pour l'établissement de l'appel et la libération des ressources dans un réseau distinct du plan d'usager qui transporte le flux de parole des communicants. Suite à cette séparation, il est mandataire alors de définir les exigences de la téléphonie en terme de sécurité et de spécifier quel plan cette architecture de sécurité tend à sécuriser. Cette solution de sécurité étant transparente au réseau de téléphonie sous-jacent, il est possible d'assurer la sécurisation des deux plans d'une façon transparente vis-à-vis l'une de l'autre en instantiant les services de sécurité de cette architecture pour chaque plan à part. Ainsi, les services de sécurité de la parole emprunteront le canal *data.*

3.6 Les différentes architectures potentielles

3.6.1 Architecture entre deux entités

Cela nécessite que les deux interlocuteurs puissent communiquer ensemble d'une manière sécurisé quelque soit le réseau de téléphonie emprunté et auquel chacun est raccordé. Il se peut que les deux soient aussi bien équipés informatiquement et dialoguent en utilisant le même logiciel et pour cela évidemment un simple micro et des hauts parleurs. Ou bien que les deux interlocuteurs utilisent une liaison de données établies en utilisant des modems via le réseau commuté (fig. 3.2) ou bien se connecter directement au réseau Internet ou Intranet de l'entreprise ou bien chacun peut être raccordé à un type de réseau de téléphonie différent que ce soit le réseau RTC, GSM ou Internet ou autre. . Aucun équipement intermédiaire est nécessaire (e.g. commutateur, routeur, passerelle, etc.).

Cette architecture ne nécessite pas l'utilisation d'une tierce entité de confiance pour gérer la sécurité entre les usagers. Les interlocuteurs déploieront un protocole de sécurité de niveau applicatif pour sécuriser leurs conversations (échanges de données transportant des échantillons de voix).

L'intérêt de ce type de communication peut se trouver dans l'application de Visioconférence, les ordinateurs se connaissent par leurs adresses IP ou bien par leurs adresses RNIS. Or, les adresses IP changeant à chaque connexion (allocation dynamique des adresses par un serveur DHCP), les correspondants doivent se mettre d'accord sur la consultation d'un annuaire (« dynamique », car mis à jour à chaque connexion par chaque correspondant potentiel qui doit s'y enregistrer) pour permettre à l'appelant de connaître l'adresse de l'appelé (cette procédure est grandement facilitée pour des utilisateurs connectés en permanence à Internet).

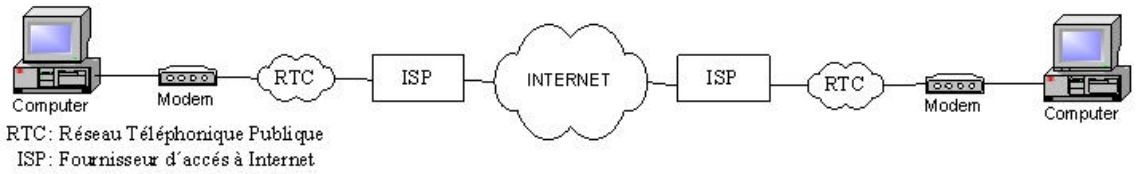

Figure 3.2 - Téléphonie entre postes informatiques.

3.6.2 Architecture entre 3 entités

L'un des correspondants se trouve derrière un équipement intermédiaire (e.g. PABX), et l'autre correspondant est sur son micro-ordinateur ; s'il désire appeler un correspondant sur le poste téléphonique de celui-ci, il doit se connecter sur un service spécial sur Internet, offert par son fournisseur d'accès à Internet (ISP pour *Internet Service Provider*), qui doit mettre en œuvre une passerelle avec le réseau téléphonique. Cela nécessite la mise en oeuvre d'une passerelle au départ et à l'arrivée de l'appel pour assurer la traduction entre les éléments spécifiques des deux réseaux (signalisation, codecs, formats de transmission d'information) afin de faire transiter la communication d'un réseau IP à un réseau téléphonique.

L'appel est taxé uniquement pour la traversée du réseau téléphonique. Ainsi, pour les appels internationaux, plus la proportion du segment IP est grande, plus l'économie réalisée sera importante.

Cette architecture nécessite l'utilisation d'une tierce entité pour gérer la sécurité entre les usagers. L'entité tierce nécessaire pour la sécurité sera alors implémentée au niveau de la passerelle ou bien du commutateur du réseau RTC ou bien au niveau du PABX. Les interlocuteurs et l'entité tierce déploieront un protocole de sécurité de niveau applicatif pour sécuriser leurs conversations (échanges de données transportant des échantillons de voix) via l'entité tierce. Cette architecture peut être utilisée quand un des interlocuteurs est sûr et n'a pas besoin d'être authentifié. On aura alors une authentification en en seul sens.

Tout comme l'architecture précédente, le poste informatique peut être relié directement au réseau Internet via une connexion à capacité de bande passante très élevée ou bien via une connexion modem (fig. 3.3).

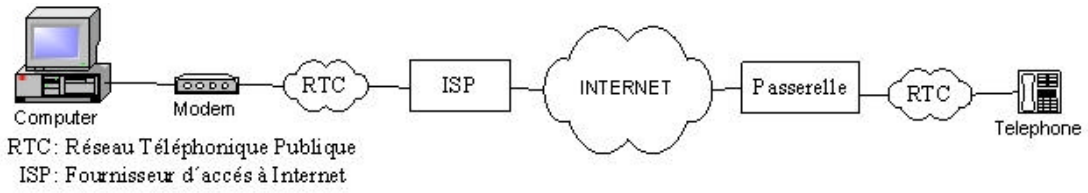

Figure 3.3 - Téléphonie entre poste informatique et téléphone.

3.6.3 Architecture entre quatre entités

Lorsque l'appelant et l'appelé sont tous les deux sur téléphone fixe et/ou mobile, et/ou bien contrôlé par un Gatekeeper (e.g. H.323) ou un Registrar (e.g. SIP) dans un réseau IP, cela nécessite la mise en oeuvre de plusieurs passerelles qui s'occupent alors de la gestion de la communication, y compris la signalisation avec le réseau téléphonique et les conversions à l'entrée et à la sortie de chaque type de réseau. Cette architecture peut être de deux types :

- soit elle est dite *hybride* (circuit / Voix sur IP). Elle est intéressante pour l'interconnexion de PABX : remplacement d'une ligne louée de type RNIS par une architecture réseau assurant une qualité de service, tout en préservant l'architecture téléphonique globale. La tarification dépend de l'opérateur ; s'il s'agit d'un réseau privé, c'est gratuit. Les téléphones classiques (non « IP-phone ») ne peuvent contacter les ordinateurs.
- Soit elle est dite *uniforme* (circuit / circuit) : elle est intéressante pour le fait d'interconnecter tout type de téléphonie fixe (e.g. RTC, RNIS) ou mobile (GSM) en mode circuit.

Cette architecture nécessite l'utilisation d'une ou plusieurs entité(s) tierce(s) pour gérer la sécurité entre les usagers. La (les) entité(s) tierce(s) nécessaire(s) pour la sécurité sera (ont) alors implémentée(s) au niveau du commutateur du réseau fixe ou mobile ou bien au niveau du PABX de l'entreprise. Les interlocuteurs et la (les) entité(s) tierce(s)

déploieront un protocole de sécurité de niveau applicatif pour sécuriser leurs conversations téléphoniques via la (les) entité(s) tierce(s).

Figure 3.4 - Téléphonie entre postes de téléphones.

3.6.4 Architecture active de la sécurité

Les réseaux traditionnels à commutation de paquet rendent capable le partage de facilités de transmission. Ceci permet un déplacement efficace des paquets de données entre les systèmes connectés. Ces réseaux définissent un modèle de communication et exécutent seulement le traitement nécessaire pour expédier les paquets vers leur destination. Actuellement, l'Internet s'est transformé d'une architecture de réseau simple à un système de livraison d'information plus sophistiqué. L'introduction de nouveaux services est une tâche difficile et demande des avancées majeures dans les méthodologies et les outils pour la création de service. Une quantité vaste de calcul spécifique pour les services, de traitement et de commutation doit être contrôlée et les nouveaux environnements de programmation de réseau doivent être organisés pour rendre possible les nouvelles infrastructures de réseau. Ces nouvelles infrastructures de réseau doivent rester ouvertes, extensible et programmable.

Cette nouvelle génération d'infrastructures de réseau est rendue possible par le concept des réseaux actifs. Ces réseaux permettent aux utilisateurs d'exécuter des codes sur les noeuds du réseau pour rencontrer les conditions spécifiques des applications. Un autre concept sera d'avoir des interfaces ouvertes bien définies au niveau des noeuds du réseau et de séparer leurs états internes des mécanismes de signalisation et de gestion. Une étude de projets de recherche sur les réseaux actifs et programmables peut être trouvée dans [Cam99] et [Ten97].

Le but des réseaux programmables est de simplifier le déploiement de nouveaux services de réseau, pour mener à des réseaux qui soutiennent explicitement le procédé de création de service et de déploiement. Les réseaux actifs représentent un saut quantique dans cette évolution. En fournissant une interface programmable dans les noeuds du réseau, ils exposent les ressources, les mécanismes et les politiques fondamentales qui sont nécessaires à ses fonctions, et fournissent des mécanismes pour construire ou raffiner les nouveaux services. Il est à noter que la programmation dans les noeuds de réseau n'est pas nouvelle. En fait, les commutateurs de téléphonie et les routeurs de réseau ne sont autres que des machines programmables et performantes, programmées pour exécuter une tâche spécifique, c.-à-d. un appel commuté ou un paquet expédié. La nouveauté dans les réseaux actifs est qu'ils révèlent cette interface programmable à l'utilisateur.

SRTP intercepte les paquets RTP, change l'entête en une nouvelle entête puis l'injecte sur le réseau pour être transporté le protocole de transport UDP et vice versa.

Cette fonctionnalité nécessite le changement au niveau des applications utilisant RTP. Afin d'éviter les changements dans les applications utilisant RTP et l'introduction de middleware nécessaire au déploiement de SRTP, nous avons proposé deux solutions pour intégrer les services de sécurité offerts par SRTP en utilisant une première méthode par les interfaces programmables et une deuxième méthode par les réseaux actifs en se basant el système d'opération Scout [Scout].

3.7 Placement des entités de sécurisation dans une architecture de téléphonie

Tout comme l'architecture de l'appel téléphonique, nous séparerons l'appel sécurisé en deux parties : une partie pour transporter le flux de parole chiffrée et une autre partie pour transporter les informations nécessaires dans des messages de signalisation propre à la solution de sécurité pour déployer l'ensemble des différents services de sécurité (i.e., les messages d'authentification des usagers).

La solution de sécurité proposée par cette architecture comprend trois acteurs principaux comme le montre la fig.3.5 : le terminal de l'usager, une entité de confiance distribuée nommé TAA (pour *Trusted Authentication Autority*) et une entité de confiance racine centralisée que nous nommerons GTAA (pour *Global Trusted Authentication Autority)*. Afin d'assurer la mobilité de l'usager, l'utilisation d'une carte intelligente (*Smart Card*) est proposée qui comprendra les paramètres de sécurité personnalisés relatifs à cette architecture.

La sécurisation des messages de signalisation entre les TAAs d'une part et entre le TAA et le GTAA de l'autre part sont basés sur des certificats. Par contre, tous les échanges entre l'usager et le TAA sont basés sur une clé pré-partagée et un identifiant unique de l'usager. La communication téléphonique sécurisée de bout en bout entre les usagers terminaux dépend d'une clé symétrique échangée entre les usagers concernés suite à la complétion de la phase d'authentification mutuelle.

L'emplacement de ses entités de sécurité dépend essentiellement du choix de l'architecture potentielle utilisée pour la sécurité de la téléphonie décrite à la section précédente. L'implémentation de cette architecture de sécurité repose sur deux choix : soit l'introduction de nouveaux composants de sécurité à l'architecture de téléphonie traditionnelle (e.g., le réseau RTC), soit l'utilisation de l'existant auquel nous ajoutons des fonctionnalités de sécurité (e.g., Gatekeeper de H.323 ou Registrar de SIP).

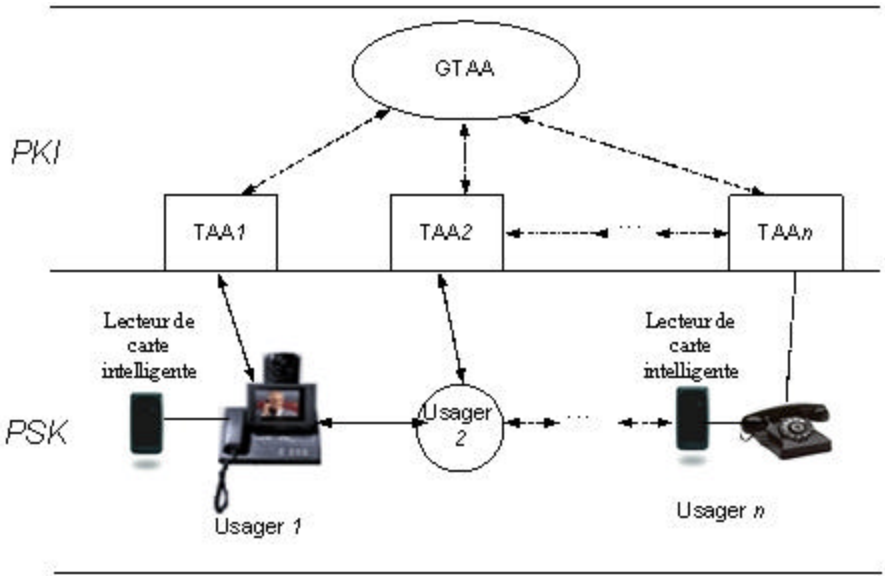

Figure 3.5 - Architecture de sécurité

3.7.1 L'entité de confiance racine

La GTAA est l'entité de confiance racine. Elle gère un groupe d'entité de confiance TAAs. Les TAA et les usagers se réfèrent à la GTAA pour résoudre le conflit de sécurité en cas de litige.

La GTAA est responsable de délivrer des certificats aux TAAs, de les sauvegarder dans une base de données public, et de communiquer avec d'autres GTAA aussi bien du réseau téléphonique traditionnel fixe, que du réseau mobile ou réseau voip. La GTAA maintient une liste de révocation des certificats qui contient la liste des certificats compromis ou expirés.

Les améliorations proposées pour la sécurité assume que chaque compagnie de téléphonie (public ou privé, PSTN/ISDN, mobile ou VoIP) établit une entité de certifications (GTAA). Les GTAA communiquent ensemble, ce qui permet au TAA ou à un téléphone individuel sécurisé dans un domaine de téléphonie spécifique puisse communiquer avec d'autres entités dans d'autres domaines d'une manière sure et sécurisée.

3.7.2 L'entité de confiance intermédiaire

La TAA est une entité de confiance qui possède des certificats pour le chiffrement et pour la signature. Chaque TAA délivre un identifiant unique et universel pour chacun de ses abonnés. Le rôle du TAA est d'authentifier les usagers, de leur livrer une clé maîtresse pré-partagée, et génère des rapports de non-répudiation signée avec son certificat de signature. Ces rapports seront fournis à l'usager. La TAA est un serveur qui peut être implémenté au niveau d'un commutateur téléphonique public, d'un PABX traditionnel, d'un PABX-IP ou au niveau du réseau GSM. La TAA gère deux bases de données, une relative aux usagers abonnés a la TAA et une autre base de données des

rapports de non-répudiation fournis par la TAA aux usagers à la fin de chaque appel bout en bout

- La base de donnée des usagers UDB. Cette base de donnée regroupe les profiles utilisateurs. Ce profile est un fichier qui contient les paramètres et les informations livrés sur la carte à puce. Le profile de l'utilisateur peut aussi donner la politique de sécurité auquel l'abonné a droit lors de son inscription avec la TAA. Cette base de donnée peut aussi contenir les profiles d'autres TAAs. Elle sera mise à jour dynamiquement. Cette partie est optionnelle et peut être géré par la TAA racine (GTAA).
- La base de donnée RDB regroupe les rapports de non répudiation générés par la TAA et signés avec le certificat de signature de la TAA. Ces rapports seront fournis aux usagers à la fin de chaque appel sécurisé authentifié et aux TAAs distants à la demande de ces dernières.

Note: ces bases de données possèdent des informations confidentielles. Il est alors crucial d'assurer un haut niveau de sécurité. La sécurisation des bases des données est en dehors du ressort de ce présent document.

3.7.3 Identifiant unique

Chaque utilisateur va s'abonner à une entité de confiance qui servira d'intermédiaire entre deux usagers communicants. Cette entité de confiance, la TAA va livrer à son abonné une carte intelligente (*SmartCard*). Cette carte contient des paramètres du protocole de sécurité dont un identifiant unique et universel.

Dans le cas où nous avons un scénario à deux usagers communicants et en absence d'entités intermédiaires, les usagers devraient obtenir d'une entité de confiance des certificats ainsi que l'identifiant. Le rôle de l'entité de confiance se restreint alors à la livraison des cartes intelligentes et des paramètres de sécurité du protocole de sécurité tel les certificats et les identifiants uniques des usagers.

Pour le calcul de l'identifiant de l'usager, la TAA utilise des informations personnelles de l'usager tel les données obtenues depuis la carte d'identité (ou *Passeport*) de l'usager, combinées à l'identité du TAA de telle façon à ne pouvoir obtenir d'un seul identifiant unique et universel pour éviter la duplication de l'identifiant entre deux usagers différents. Cet identifiant sera signé par l'entité de confiance de racine, la GTAA. Il est à noter que cet identifiants, les paramètres de sécurité du protocole et l'API peuvent être fourni à l'usager sous forme de « *patch* » à obtenir d'un serveur et à installer sur leur téléphone ayant les capacités en ressource nécessaire pour supporter le protocole et ses algorithmes de sécurité. Toutefois, pour assurer la mobilité des usagers et pour éviter les attaques susceptibles de se produire durant la phase de récupération du « *patch* », ces informations seront livrées directement à l'usager.

3.7.4 Les cartes intelligentes

Les cartes intelligentes ont longtemps été associées à la sécurité puisqu'elles fournissent une solution partielle au besoin d'identification personnelle et de non-répudiation. Dans la mesure où une carte est résistante aux manipulations, elle peut être

utilisé pour emmagasiner des secrets importants tels que clés de chiffrement et les identifiants des usagers.

Cette carte à puce sera utile et nécessaire pour les raisons suivantes:

- Eviter de récupérer l'identifiant, les paramètres de sécurité et l'API de sécurité depuis un serveur. Nous éviterons ainsi les attaques par homme de milieu en gardant l'identifiant de l'usager sécurisé et privé.
- Elle peut être utilisée aussi bien sur des téléphones publics ou privés, fixes ou sans fils, IP hardphone ou softphone.
- Elle est nécessaire pour la mobilité de l'usager.
- Parmi les paramètres de sécurité que contient la carte intelligente, nous citerons une clé pré-partagée entre l'usager et la TAA, les certificats du TAA pour valider la clé pré-partagée ou ceux de l'usagers en absence d'entités intermédiaires, une fonction de hachage, etc.

3.8 Définition des politiques et leur impact sur les architectures

La sécurité de la téléphonie est un tout. Le synoptique suivant [Aut02] positionne les différentes briques qui vont contribuer à la mise en œuvre d'une bonne sécurité. Le responsable de sécurité du système téléphonique (e.g. opérateur de téléphonie, entreprise ou particulier) est responsable de la définition des exigences affectées à chacune de ces briques et de leur mise en cohérence. L'ensemble est chapeauté par la politique de sécurité téléphonique.

La politique de sécurité d'un système de téléphonie est un ensemble de lois, règlements et pratiques qui régissent la façon de gérer, protéger et diffuser les biens, en particuliers les informations sensibles entre les entités concernées. On dispose actuellement de catalogues d'énoncés de règles qui sont reconnus par les spécialistes du domaine de sécurité [ISO 17799].

La politique de sécurité est une déclaration des restrictions sur l'accès aux objets et/ou au transfert d'information, ou plus généralement la déclaration formelle de l'authenticité des entités communicantes, la confidentialité du système, la disponibilité, ou les conditions d'intégrité. La politique de sécurité applique la condition de "séparation des politiques et des mécanismes de sécurité".

Cette politique va définir les services de sécurité qui seront requis pour répondre aux besoins des utilisateurs et, afin de rendre ces services, l'opérateur de téléphonie va développer les mécanismes de sécurité afférents. Ces mécanismes vont utiliser des informations de sécurité comme des clés cryptographiques ou des algorithmes. A leur tour, ces informations vont devoir être gérées avec le plus grand soin et pour cela le système s'appuiera sur des ressources de gestion des fonctions de sécurité.

Les services de sécurité vont s'appuyer sur différents mécanismes qui pourront être utilisés seul ou en combinaison. Les mécanismes de sécurité vont être appliqués aux données (e.g. échantillon de parole ou signalisation d'appel) traitées par les applications de téléphonie et combinées à des informations de sécurité, dont les clés cryptographiques. Ces informations de sécurité vont à leur tour être gérées pendant tout leur cycle de vie par les infrastructures de gestion des fonctions de sécurité.

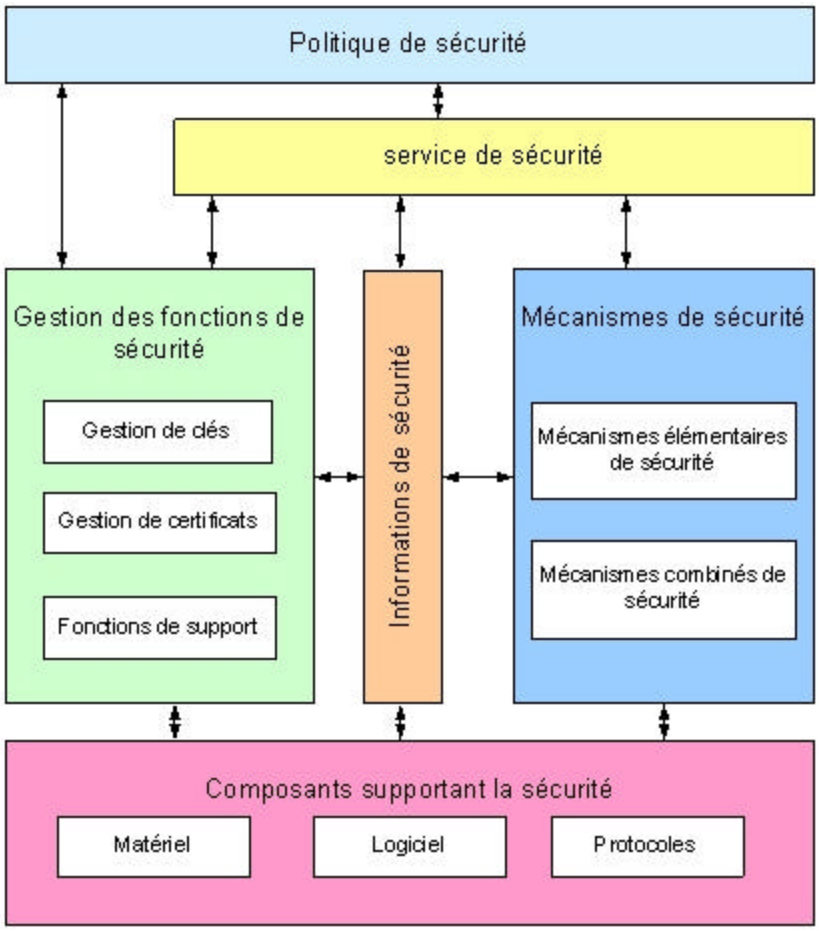

Figure 3.6 - synoptique des ressources de sécurité

Les composants qui supportent la sécurité sont, à l'instar des autres ressources, formés d'éléments matériels, de logiciels et de protocoles d'échanges. Puisque cette architecture de sécurité utilise des certificats et des clés pré-partagées, il est impératif d'utiliser des dispositifs matériels spécifiques pour la protection des clés cryptographiques. En effet, la divulgation de certains secrets pourrait rendre caduc l'ensemble de l'édifice de sécurité. Pour cet effet, il faut disposer d'équipements pour les TAA et les GTAA dotés de protections passives et actives, si possible ayant subi des évaluations de résistance et certifiés par un organisme officiel. D'autre part, il faut les localiser dans des locaux eux-mêmes placés sous protection active.

Un *framework* de sécurité permet de définir un guide de recommandations de haut niveau ainsi que, dans un deuxième temps, des guides et règles. Chaque organisme (e.g. opérateur, fournisseur de service, entreprise, individu) peut définir son *framework* de sécurité [Ced03] et ses propres politiques de sécurité, à condition de tenir compte des spécificités et besoins de sécurité. Nous définirons les quelques modèles suivants de politiques de sécurité mais nous nous ne limiterons pas à ces modèles:

- Politique libre :

  Cette politique ne nécessite aucun niveau de sécurité et donc l'appel téléphonique passe en clair entre les entités communicantes. Aucune sécurité suivant cette architecture n'est assurée aux usagers de téléphonie.

- Politique sûre de l'infrastructure :

Cette politique requiert l'authentification de l'infrastructure du réseau vis-à-vis de l'usager. Donc elle nécessite une authentification du TAA par l'usager et vice versa.

- Politique sûre du distant :

Cette politique est similaire à la précédente mais l'entité à authentifier peut être aussi bien l'usager distant ou la TAA auquel l'usager d'origine est rattaché.

- Politique sûre du distant avec chiffrement :

Cette politique contribue à authentifier l'entité distante avec possibilité de négocier certains services de sécurité avec le réseau sous-jacent (e.g. possibilité d'utiliser SRTP dans un contexte VoIP pour sécuriser le flux média). Nous pouvons alors authentifier les usagers avec le protocole de sécurité et chiffrer avec un protocole de sécurité du réseau sous-jacent (e.g. SRTP si l'appel se déroule dans un contexte voix sur IP).

- Politique sûre du distant avec chiffrement de bout en bout :

Cette politique impose l'application de tous les services de sécurité offerts par le protocole de sécurité.

3.9 Les contrats de service

Un SLA (pour *Service Level Agreement*) [Wus02] est un contrat établi entre le fournisseur de service et le client. Le fournisseur de service peut être un opérateur de télécommunication fixe ou mobile, un fournisseur d'accès Internet (et donc de VoIP), une compagnie privée (e.g. une entreprise). Le service offert sera alors la sécurisation de l'appel téléphonique du client fournit par le fournisseur de service. Le SLA spécifie les termes de l'accord et combien le client doit payer pour ses services (si ce service est payant).

Un bon SLA adresse cinq aspects clés :

- Que promet le fournisseur.
- Comment le fournisseur livrera ces promesses.
- Qui mesurera la livraison, et comment.
- Qu'est ce qui arrive si le fournisseur ne livre pas le service comme promis.
- Comment le SLA changera progressivement.

Le défi pour un nouveau service et son SLA associé est qu'il y a une relation directe entre l'architecture et quels sont les niveaux maximums de disponibilité du service de sécurité. Ainsi, un SLA ne peut pas être créé le vide. Un SLA doit être défini avec l'infrastructure de téléphonie à l'esprit.

Une relation exponentielle existe entre les niveaux de disponibilité et le coût apparenté. Quelques clients ont besoin des plus hauts niveaux de disponibilité et veulent payer plus. Ainsi, différents SLAs avec des coûts associés est une approche commune pour satisfaire les besoins des clients.

Comme les services et les technologies changent, le SLA peut changer pour refléter les changements de et/ou d'améliorations. Ce SLA sera réexaminé chaque six mois et sera mis à jour comme nécessaire. Quand les mises à jour sont estimées nécessaire, la clientèle sera demandée de réexaminer et approuver les changements.

D’autres secteurs doivent être définis et détaillés dans un SLA. Ces détails portent sur les points suivants :

- comment les mesures sont faites,
- quelle limitation d'usage du service si plusieurs utilisateurs demandent le même service simultanément, etc.
- détails sur comment et qui reçoit des rapports et comment les conflits sont arbitrés.
- Quelles sont les politiques de sécurité requises par le client et offertes par le fournisseur de service de sécurité.

3.10 Les définitions des services de bout en bout

La section 3.4 de ce chapitre spécifie les services de sécurité majeurs nécessaires à la téléphonie et qui sont au nombre de 5 : l’authentification de l’usager, la confidentialité, l’intégrité, le non-rejeu et la non-répudiation.
Si nous voulons catégoriser ces services, nous les séparerons en deux catégories :

- catégorie des services de bout en bout, notamment :
  - l’authentification mutuelle des usagers ;
  - la confidentialité des données de parole et de signalisation ;
  - l’intégrité ;
  - la non-répudiation de l’appel.
- catégorie des services locaux, notamment :
  - l’authentification locale de l’usager à une entité de confiance ;
  - le non-rejeu.

3.11 API pour la sécurisation de la téléphonie

La caractéristique principale de cette architecture de sécurité est d’offrir un cadre sécuritaire à la téléphonie indépendante de l’infrastructure du réseau sous-jacent. Cette indépendance introduit la séparation des rôles entre l’application de sécurité d’une part et les applications et les plates-formes de téléphonie d’autre part. Les capacités de l’application de sécurité sont séparées des services offerts par les opérateurs de téléphonie. Des interfaces normalisées permettront d’accéder aux ressources des réseaux et d’instancier les services de sécurité.

L’offre des services de sécurité doit être indépendante des réseaux. Ces services doivent donc être portables et adaptables aux réseaux de téléphonie (RTC, RNIS, GSM, UMTS, VoIP, etc.) et aux terminaux (fixe, mobile, PC, PDA, etc.). Des interfaces de programmation de l’application de sécurité devraient être spécifiées dans cette perspective. Elles permettent de développer les services de sécurité proposée par cette architecture en se basant sur l’API de sécurité multiréseaux et multiterminaux, en offrant à l’utilisateur un accès sécurisé à son environnement de téléphonie.

Dans cette optique, l’API de sécurité (SAPI) permet de spécifier les interfaces entre la couche service de sécurité et la couche réseau permettant de mapper le service de sécurité sur tout type d’infrastructure de téléphonie.

Le point où cette interface est localisée détermine le degré de contrôle que cette application possède. Dans un scénario de contrôle d'appel sécurisé de premier-parti, l'interface est localisée au niveau du terminal. L'application possède un contrôle d’appel

initié par l'utilisateur téléphonique. Dans un scénario de contrôle d'appel de tiers l'interface est localisée dans le système téléphonique. Cette application aura des capacités différentes que ceux dans un scénario de contrôle d'appel sécurisé de premier-parti. Un troisième modèle consiste à distribuer l'application de sécurité entre le terminal et l'entité tierce. L'API de sécurité aura des capacités différentes sur chaque parti.

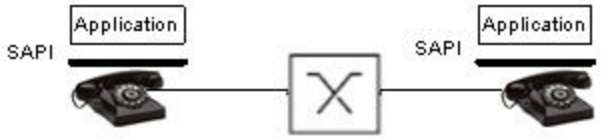

Figure 3.7 - scénario de contrôle d'appel sécurisé de premier-parti

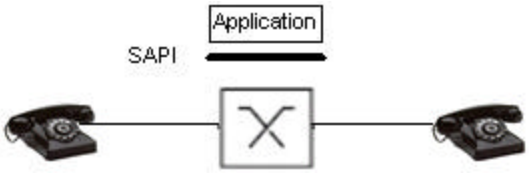

Figure 3.8 - scénario de contrôle d'appel sécurisé de tiers

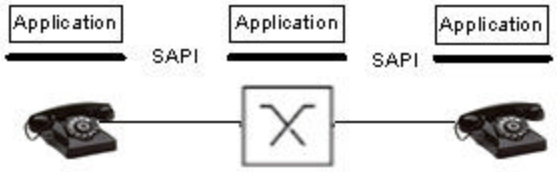

Figure 3.9 - scénario de contrôle d'appel sécurisé de premier parti et de tiers

3.11.1 Le modèle d'appel SAPI

3.11.1.1 Principes

La conception de SAPI est basée sur un modèle d'appel qui présente un haut niveau d'abstraction et une indépendante de la technologie des réseaux au-dessus desquels SAPI va tourner. Il décrit l'appel comme étant un ensemble de machines à états finis (FSM pour *Finite State Machine*) qui subissent des transitions d'un état à un autre état selon l'évolution de l'appel sécurisé.

Le but du modèle d'appel est d'être très général pour couvrir beaucoup de scénarios d'appel différents. Ainsi, il serait capable de décrire par exemple :

- Un appel entre deux partis,
- De multiples appels simultanés effectués sur le même terminal,
- Une conférence entre les partis multiples,

Le modèle d'appel décrit aussi bien l'appel sécurisé que les partis impliqués dans l'appel.

Le modèle d'appel définit cinq classes de base (dans le sens général de types d'objet). Deux classes décrivent les partis impliqués dans l'appel. Leurs objets sont persistants et indépendants de l'appel sécurisé :

- Un utilisateur est représenté par un objet *SecureAdresse*. L'attribut principal de l'objet *SecureAddress* est l'identifiant de l'utilisateur.
- Un terminal téléphonique est représenté par un objet *SecureTerminal*. L'attribut principal de l'objet *SecureTerminal* est l'adresse du terminal.

Les trois autres classes décrivent un appel sécurisé. Leurs objets ne sont pas persistants, mais ils sont créés dynamiquement pendant un appel sécurisé. Chaque objet inclut une machine à états finis :

- Un objet *SecureCall* est créé pour chaque appel sécurisé.
- Un objet *SecureConnection* est créé pour chaque utilisateur participant dans l'appel sécurisé. Il connecte l'objet *SecureAddress* de l'utilisateur avec l'objet Appel de l'appel sécurisé.
- Un objet *TerminalSecureConnection* est créé pour chaque terminal participant à l'appel. Il connecte l'objet Terminal du terminal avec l'objet Connexion.

3.11.1.2 Exemple d'un appel sécurisé basé sur le modèle d'appel

Cette section présente un exemple d'un appel sécurisé choisi entre deux participants pour aider à mieux expliquer le modèle d'appel. Cet exemple est montré dans la Figure 3.10. Il y a deux objets de *SecureConnexion* attachés à l'objet *SecureCall*, un pour chaque participant. Cette représentation est importante pour l'extension de l'appel à un appel conférence avec trois ou plusieurs partis. Il est à noter que le modèle est complètement symétrique, il ne distingue pas entre les entités locales et éloignées.

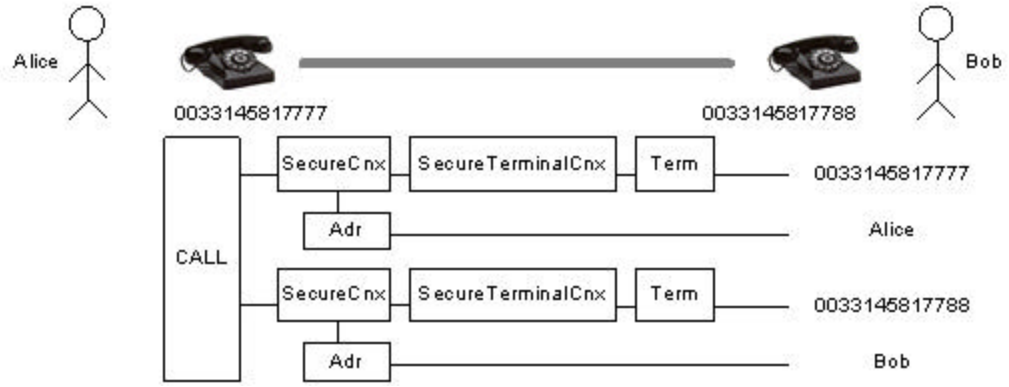

Figure 3.10 - modèle d'appel sécurisé à deux partis

3.11.1.3 Machines à états finis

Cette section décrit les objets apparentés à l'appel sécurisé et leurs machines à états finis avec plus de détail.

Objet *SecureCall*

Un objet *SecureCall* est créé pour chaque appel. L'état de l'objet *SecureCall* dépend du nombre d'objets *SecureConnexion*.

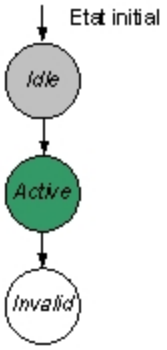

Figure 3.11 - Machine à état de l'objet *SecureCall*

IDLE C'est l'état initial pour tous les objets *SecureCall*. Dans cet état, *SecureCall* n'est relié à aucun objet *SecureConnexion*.
ACTIVE un objet *SecureCall* avec une certaine activité passe à cet état. Les objets *SecureCall* associés à un ou des objets *SecureConnexion* doivent être dans cet état.
INVALID c'est l'état final par lequel passe touts les objets *SecureCall*. Les objets *SecureCall* qui perdent tous leurs objets *SecureConnexion* (via une transition de l'objet *SecureConnexion* à l'état *SecureConnection.Disconnect*) passeront à cet état. Ces objets *SecureCall* ne possèdent aucune *SecureConnection* et ne peuvent pas être utilisés pour une action future.

Objet *SecureConnection*
Un objet *SecureConnexion* est créé pour chaque utilisateur participant dans l'appel. Il connecte l'objet *SecureAddress* de l'utilisateur avec l'objet *SecureCall.*

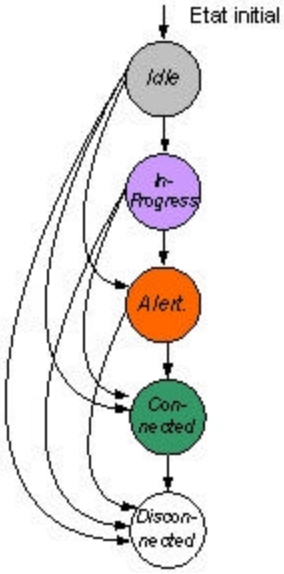

Figure 3.12 - machine à état fini pour l'objet *SecureConnection*

IDLE C'est l'état initial pour toutes les nouvelles *SecureConnexion*. Les connexions qui sont dans l'état de *SecureConnexion.Idle* ne font pas partis activement à un appel téléphonique sécurisé, cependant leurs références aux objets *SecureCall* et *SecureAddress* sont valides pour un appel non sécurisé.
INPROGRESS Cet état implique que la *SecureConnexion*, qui représente la destination d'un appel téléphonique sécurisé, passe au processus de contacter le destinataire et donc de négocier avec lui les paramètres de sécurité dont la première phase est l'authentification des usagers.
ALERTING Cet état implique que l'objet *SecureCall* est notifié qu'un appel sécurisé entrant arrive.
CONNECTED Cet état indique qu'une *SecureConnection* et son *SecureAddress* sont impliqués activement dans un appel téléphonique. En d'autres termes, deux personnes communicantes d'une manière sécurisées seront représentées par deux *SecureConnection* dans l'état *SecureConnection.Connected*.
DISCONNECTED Cet état indique qu'une *SecureConnection* n'est plus impliquée dans l'appel téléphonique, bien que ses références pour *SecureCall* et Secure*Address* restent valides. Une *SecureConnection* dans cet état est interprétée comme avoir appartenu précédemment et pour une fois à cet appel téléphonique. Cet état permet à l'utilisateur de repasser en mode non sécurisé.
FAILED Cet état indique qu'une *SecureConnection* a échoué pour une raison quelconque, qui peut être que la négociation des paramètres de sécurité n'a pas abouti ou bien que l'un de correspondant n'est pas authentique.
Dans la machine à état de l'objet *SecureConnection* ci-dessous, les états *Disconnected* et *Failed* ne figurent pas.

Objet *SecureTerminalConnection*
Un objet *SecureTerminalConnection* est créé pour chaque terminal téléphonique participant dans une *SecureConnexion*. Il connecte l'objet *SecureTerminal* du terminal avec l'objet *SecureConnexion*.

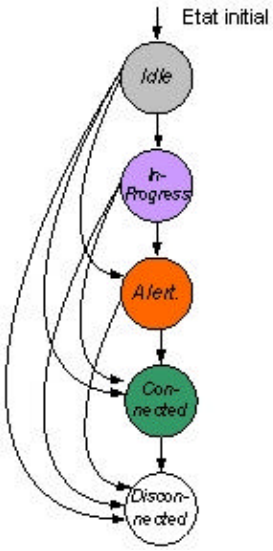

Figure 3.13 - machine à état fini pour l'objet *SecureTerminalConnection*

IDLE Cet état est l'état initial pour tout état *SecureTerminalConnection*. Les objets de *SecureTerminalConnection* ne restent pas dans cet état pour longtemps. Ils transitent typiquement dans un autre état rapidement pour établir l'appel sécurisé.
RINGING Cet état indique qu'un *SecureTerminal* reçoit une connexion sécurisée venant de l'extérieur.
ACTIVE Cet état indique qu'un *SecureTerminal* est actif dans un appel téléphonique sécurisé. Ceci implique d'habitude que la personne utilisant ce *SecureTerminal* fait partie de l'appel téléphonique sécurisé.
PASSIVE Cet état indique qu'un *SecureTerminal* fait partie d'un appel téléphonique sécurisé mais pas dans un mode actif. Ceci peut impliquer qu'une ressource du *SecureTerminal* est utilisée et peut limiter les actions sur le *SecureTerminal.* Cet état peut indiquer que le SecureTerminal attend un paramètre de sécurité (comme la fin d'un appel sécurisé ou bien la génération du rapport de non-répudiation).
DROPPED Cet état indique qu'un *SecureTerminal* particulier a quitté d'une façon permanente l'appel téléphonique sécurisé. Il peut toujours participer à un appel téléphonique non sécurisé.

3.11.2 Les fonctions de sécurité de cette architecture

Les fonctions de sécurité qui contrôlent des données sensibles (e.g. voix) imprègnent l'architecture de sécurité. Ceci implique que ces besoins de sécurité doivent être considérés dans chaque aspect de la conception de l'architecture, et que ces aspects de sécurité doivent être conçus dès le début parce qu'il est très difficile de verrouiller les lacunes de sécurité par la suite. Une référence standard [Sal75] recommande qu'une architecture de sécurité ait les propriétés énumérées ci-dessous, avec les annotations expliquant les approches qui seront utilisées surtout dans la phase d'implémentation de cette architecture:

- accès basé sur les permissions: Les permissions implicites d'accès/usage doivent être un accès libre (sans sécurité), avec les droits d'accès ou usage des services de sécurité étant sélectivement disponible selon les politiques de sécurité exigées. Si nous considérons ses services de sécurité comme étant des objets, il seront visibles au processus qui les à instancier. Ceci facilite l'utilisation de cette architecture de sécurité. Pour cette raison, l'accès libre laisse le choix à l'utilisateur de spécifier quels services de sécurité il requiert en sélectionnant une politique de service ajustée à ses besoins. Politique de sécurité sera spécifiée dans le contrat de service de sécurité établi avec le fournisseur de service de sécurité.

- Moins de privilège et d'isolement : chaque objet (e.g. service de sécurité) doit opérer avec le moindre privilège possible pour minimiser les dommages dus à des attaques malicieuses. Les objets doivent rester séparés pour réduire le compromis des informations transportées involontairement ou délibérément ou bien le compromis des capacités échangées (e.g. paramètres de sécurité). Ces deux conditions vont de paires puisque chaque objet aura un accès minimal aux ressources requises pour performer cette tâche. Un exemple, si un objet-certificat possède un objet de cryptage attaché à lui, l'objet-cryptage pourra être utilisé d'une manière cohérente avec les attributs positionné dans le certificat-objet (il

pourra être utilisable seulement pour la vérification de signature, mais pas pour le cryptage ou l'échange clé, ou pour la génération d'une nouvelle clé pour l'objet).

- Compléter la médiation : l'accès à chaque objet est vérifié à chaque fois que l'objet sera utilisé. Il n'est pas possible d'accéder à cet objet sans l'avoir vérifié au préalable.

- Conception ouverte: l'architecture de sécurité désignée pour protéger les échanges de la téléphonie doit être la plus simple possible pour permettre d'être testée et vérifiée facilement, fiable et ne doit pas compter sur la sécurité par l'obscurité.

- Facile à utiliser: Afin de promouvoir son usage, l'architecture de sécurité doit être facile à utiliser et transparente autant que possible à l'utilisateur. Dans presque tous les cas, l'utilisateur n'est pas même conscient de la présence des fonctions de sécurité. L'architecture emploie un noyau de sécurité pour appliquer ses mécanismes de sécurité. Ce noyau fournit l'interface entre le réseau téléphonique sous-jacent et les objets de l'architecture et entre les objets eux-mêmes. Les fonctions de sécurité sont contenues dans le noyau de sécurité pour les raisons suivantes [PFL03]:
    - Séparation: en isolant les mécanismes de sécurité de l'ensemble de l'implémentation, il est plus facile alors d'assurer la protection contre les manipulations et les pénétrations du système de téléphonie.
    - Unité: toutes fonctions de sécurité sont exécutées par un seul module de code.
    - Modifiable: Les changements au mécanisme de sécurité sont plus faciles à faire et essayer.
    - Caractère compact : Parce qu'il exécute seulement des fonctions de sécurité, le noyau de sécurité va être petit en toute probabilité.
    - Couverture: Chaque accès à un objet protégé est vérifié par le noyau.

3.12 Validation par un scénario abstrait de cette architecture

3.12.1 Analyse du modèle d'appel et de l'architecture

Pour pouvoir proposer une architecture de sécurité aux communications de voix, nous allons commencer par définir les différents modèles d'appels utilisés en téléphonie. En principe, pour acheminer des données entre deux usagers, une communication est établie entre les deux entités comme suit :

- un (des) canal (aux) (physique ou logique) est (sont) établi(s) entre deux usagers qui sera (seront) utilisé(s) pour acheminer les échantillons de voix pour toute la durée de la communication. Nous pouvons donner en exemple deux usagers connectés entre eux par le réseau téléphonique commuté/réseau GSM, ou connectés entre eux à travers le réseau Internet pour un appel de voix sur IP ;
- une communication est établie entre les deux usagers distants en passant au préalable à travers un ou plusieurs équipements intermédiaires auquel (auxquels) au moins un des deux communicants est connecté. Cet équipement intermédiaire peut

être un PABX conventionnel interne d'une entreprise ou bien une passerelle de voix sur IP.

Afin de sécuriser l'appel de voix entre deux communicants, il faut s'assurer du modèle d'appel utilisé et des vulnérabilités relatives aux infrastructures mises en œuvre pour effectuer ce modèle de communication.

Comme un appel peut passer au travers d'infrastructures hétérogènes, il est alors nécessaire de faire abstraction de l'infrastructure sous-jacente et d'assurer une sécurisation de la communication indépendamment des techniques déployées pour établir l'appel.

Les aspects de sécurité entrent en jeu lorsqu'il est nécessaire ou préférable de protéger l'information transmise d'un adversaire qui pourrait menacer la confidentialité, l'authenticité, l'intégrité, etc. Toutes les techniques de sécurité ont deux composantes :

- une transformation relative à la sécurité de l'information à envoyer par chiffrement des messages ;
- une information secrète partagée par les deux acteurs, et de préférence, inconnue d'un éventuel adversaire. Une clé de chiffrement utilisée pour le brouillage des messages à l'émission et leur décodage à la réception.

Une tierce partie de confiance peut s'avérer nécessaire pour réaliser une transmission sûre. Elle peut, par exemple, être responsable de la distribution de l'information secrète aux deux acteurs tout en la préservant de toute agression. Elle peut jouer le rôle de l'arbitre entre deux acteurs en dispute concernant l'authenticité et la non-répudiation de la communication.

Des politiques de sécurité qui gère les mécanismes de sécurité à appliquer à un appel suivant les nécessités et les requis du type de l'appel doivent être aussi définies.

3.12.2 Scénario abstrait de l'appel

L'appel téléphonique se compose de deux canaux : un canal pour la signalisation de l'appel et un canal de donnée pour transporter la conversation des usagers.

La figure 3.14 donne un scénario abstrait d'un appel téléphonique sécurisé. L'appel est établi et terminé en mode natif suivant les procédures d'ouverture et de fermeture du protocole de téléphonie appliqué dans le réseau sous-jacent. Le canal de donnée est sécurisé selon la solution décrite par cette architecture de sécurité. Nous avons représenté dans cette figure trois services de sécurité en utilisant le modèle avec entité tierce de confiance. Les services de sécurité sont l'authentification des usagers à travers une entité de confiance, la confidentialité et le rapport de non-répudiation. Chaque usager impliqué dans la communication envoie une requête d'authentification à l'entité de confiance lequel l'authentifie localement. Une fois les deux usagers se sont authentifiés localement avec l'entité de confiance, cette dernière confirme l'authentification des usagers de bout en bout. S'ensuit alors la phase de chiffrement du flux de parole. Nous considérons que le service d'intégrité est implicite avec l'authentification et la confidentialité. Le non-rejeu étant un service à traitement local, il n'est pas représenté ici. Une fois l'appel sécurisé est terminé, un rapport de non-répudiation est généré par l'entité de confiance et distribué aux usagers.

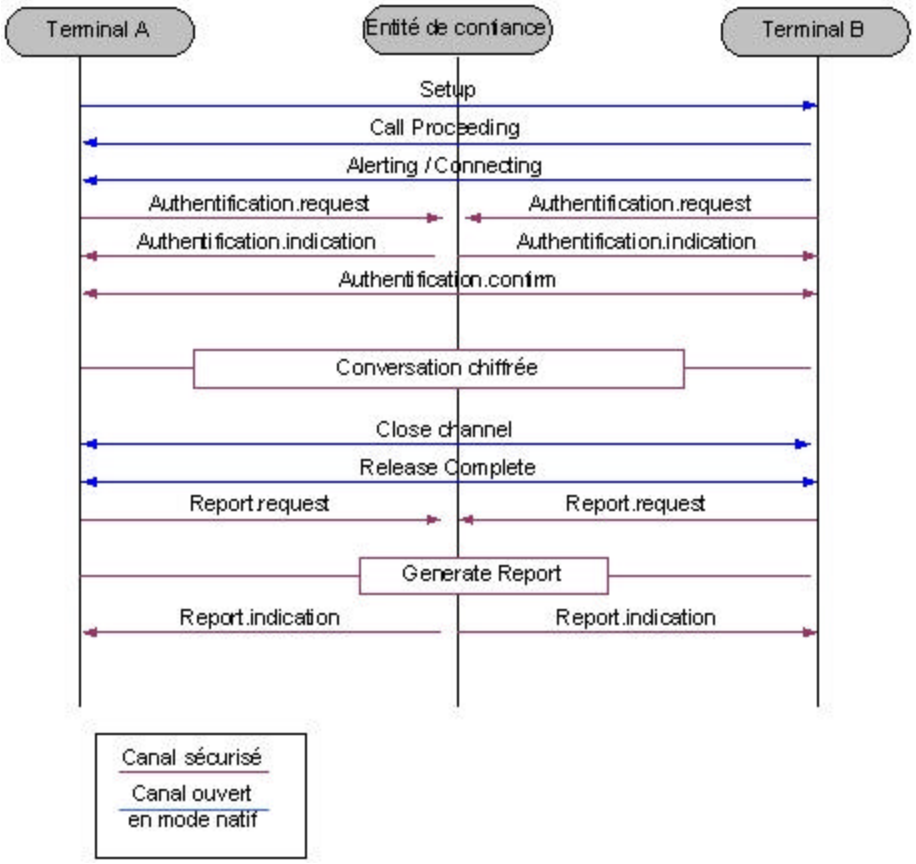

Figure 3.14 - scénario abstrait d’un appel téléphonique sécurisé

3.13 Les caractéristiques de la conception de cette architecture

Tout comme l'architecture de logiciel est basée sur plusieurs buts de conception, cette architecture de sécurité est aussi construite sur plusieurs principes spécifiques. Ceux-ci sont :

- La séparation des politiques et des mécanismes de sécurité : Le composant de politique traite des décisions spécifiques du contexte des objets et exige la connaissance détaillée de la sémantique de chaque type d'objet. Le mécanisme traite l'implémentation et l'exécution d'un algorithme pour appliquer la politique. Le contexte et l'interprétation exacte sont fournis en apparence par le composant de politique. En particulier c'est important que la politique ne soit pas codé dans le mécanisme d'exécution, comme c’est le cas pour plusieurs systèmes (e.g., Bookbased Orange [Orange]). L'avantage de cette forme de séparation est qu'il devient alors possible de changer la politique pour convenir des applications individuelles sans exiger la réévaluation du système entier

- Conception vérifiable : il doit être possible d'appliquer les techniques formelles de vérification à la portion critique de sécurité de l'architecture (le noyau de

sécurité) afin de s'assurer que les mesures de sécurité sont appliquées comme voulu (ceci est une condition de Livre Orange [orange] standard pour les noyaux de sécurité, bien que rarement atteint). De plus, ce doit être possible d'exécuter cette vérification complètement jusqu'au code.

- Une politique flexible de sécurité : Puisque tous les utilisateurs n'exigent pas des politiques de sécurité pareilles, ce doit être relativement facile à adapter les détails de la politique de sécurité aux conditions spécifiques des utilisateurs sans beaucoup d'effort de la part de l'utilisateur ou bien sans le besoin de réévaluer le système entier quand un changement de politique mineur est effectué.

- Une implémentation efficace et simple : une implémentation efficace et robuste du noyau de sécurité doit être un but primaire de conception pour l'architecture de sécurité. Le but primordial est s'assurer que le noyau fournit un niveau supérieur d'exécution, extrêmement simple, efficace et facile à vérifier dans la mesure où l'utilisateur n'est pas même conscient de la présence de ce noyau.

3.14 Conclusion: les avantages et inconvénients de cette solution

Le nouveau caractère de sécurité de cette architecture utilisant des cartes intelligentes promet d'aider à résoudre quelques problèmes de sécurité importants adressés dans le monde de la téléphonie fixe, mobile et IP. Mais l'utilisation des cartes intelligentes apporte aussi de nouveaux risques. Le dilemme de la sécurité reste comme suit : quel degré de risque à prendre, par contre quelles seront les bénéfices qui en découlent?

L'utilisation des certificats ou des clés PSK avec les cartes intelligentes rend la sécurité vulnérable si les clés sont portées à l'extérieur de la carte pour effectuer la signature digitale ou pour le chiffrement/déchiffrement. Actuellement, il est possible avec les nouvelles technologies des cartes à puce de déployer des fonctions de sécurité enfoncées dans la carte elle-même, ceci éviterait d'exporter les clés privées et publiques en dehors de la carte.

La définition des politiques de sécurité permet à l'usager et à l'entité de confiance (TAA) une plus grande flexibilité. Ainsi, chaque usager a le choix des services de sécurité dont il a besoin. Ces politiques de sécurité sont accompagnées par un contrat de service (SLA) qui sera établi entre l'usager et le TAA.

Chapitre 4

4 Le protocole SVSP (Simple Voice Security Protocol)

4.1 Résumé

Dans ce chapitre, nous proposons de présenter et de détailler le protocole de sécurité SVSP qui permet la sécurisation de la téléphonie et qui se base sur l'architecture de sécurité définie précédemment. Ce protocole permet de sécuriser l'appel de bout en bout indépendamment de l'infrastructure du réseau sous-jacent. Il est inspiré des techniques de sécurité présentées par les protocoles de sécurité existants.

4.2 Motivations

Dans le monde de la téléphonie, chaque type de réseau possède ses propres mécanismes et protocoles pour sécuriser les échanges entre les entités communicantes. Comme le montre le chapitre 3, différentes solutions de sécurité sont présentes dans les réseaux VoIP et GSM alors que la sécurité dans d'autres est quasi-inexistante. Toutefois, la NSA a défini un protocole de sécurisation de la téléphonie, le FNBDT, faisant abstraction de l'infrastructure sous-jacente. Ce protocole permet une interopérabilité dans les communications sécurisées de la voix traversant plusieurs plates-formes interconnectées. Cependant cette solution reste inaccessible aux particuliers. C'est une solution considérée classifiée par la NSA et est sous l'emprise du pouvoir et du monopole des gouvernements des grands pays comme l'Amérique du Nord, les pays du *NATO* et quelques autres grandes nations.

Dès lors, notre but est de fournir une solution de sécurisation des échanges téléphoniques entre les entités communicantes et qui soit accessibles aux particuliers et aux entreprises. Elle peut être tout aussi bien adoptée par les grands opérateurs et les fournisseurs d'accès.

Notre solution se base sur la conception d'un protocole de sécurité, le *Simple Voice Security Protocol*. De niveau applicatif, il définit des règles d'échanges et les paramètres de sécurité nécessaires aux déploiements des mécanismes de sécurisation. Le protocole SVSP assure les services de sécurité nécessaires à la sécurisation de la téléphonie tels l'authentification de l'usager, la confidentialité, l'intégrité, le non rejeu et la non-répudiation de l'appel. SVSP met en relief essentiellement deux services qui sont l'authentification et la non-répudiation.

Nos motivations pour la définition de SVSP sont les suivantes :

1. *Supporter l'authentification mutuelle et la protection d'identité des deux communicants*

L'absence d'authentification solide des entités communicantes dans la téléphonie laisse un risque de substitution d'identité et ainsi, l'accès illicite aux communications. Ainsi un malfaiteur peut voler l'identité d'une entité communicante et peut se présenter comme étant cette dernière. Pour palier à cette attaque, l'authentification de l'usager est nécessaire et peut se baser sur un mécanisme utilisant une dérivée de son identifiant. En effet, l'usager possèdera un identifiant unique bien protégé qui lui permet de s'authentifier auprès d'une entité tierce de confiance ou auprès de l'entité communicante distante. Avec SVSP, ce service concerne surtout l'authentification de l'usager auprès de l'entité distante et la protection de l'identité de l'usager puisqu'un serveur SVSP, fournisseur des identifiants uniques, protège cette identité par une clé symétrique non partagée.

2. *Support des certificats d'identités et de clés pré-partagées*

Pour assurer l'authentification des usagers, deux méthodes seront utilisées. La première méthode consiste a utilisé des certificats pour chiffrer et signer les messages échangés pour effectuer l'authentification des usagers. Pour éviter les lourdes opérations de chiffrement et de signature par un couple de clés public/privée, la deuxième méthode pousse l'usager à partager avec une entité tierce de confiance une clé symétrique. Cette clé sera utilisée pour chiffrer les messages d'authentification et pour générer les condensâts pour assurer l'intégrité des message.

3. *Interopérabilité des communications sécurisées entre différentes infrastructures de téléphonie*

La première motivation derrière la conception du protocole SVSP est le développement d'un protocole qui permet aussi bien une sécurisation de l'appel téléphonique. La deuxième motivation, celle qui est la plus importante, est que SVSP assure une interopérabilité de cet appel sécurisé établi entre les différentes infrastructures de réseau de téléphonie. Ceci permet :

a. Sécuriser l'appel de bout en bout indépendamment de l'infrastructure du réseau sous-jacent.
b. Eviter de modifier l'infrastructure du réseau
c. Puisque SVSP est de niveau applicatif, ceci permet de définir une API SVSP qui jouera le rôle d'intermédiaire entre SVSP et le protocole de communication appropriée à chaque type de réseau de téléphonie.
d. Le protocole SVSP permet de négocier les paramètres de sécurité entre les entités communicantes. Ceci permet de limiter certaines fonctionnalités redondantes au sein de chaque protocole de sécurité (authentification, intégrité, etc.).
e. De simplifier le transfert d'un protocole à un autre et réduire les temps d'établissement d'un canal sécurisé par chaque protocole. Chaque protocole définit seulement son échange en phase 2.

4.3 Les étapes de conception du protocole SVSP

Concevoir un nouveau protocole nécessite la détermination des besoins et la définition des exigences qui seront couverts par ce protocole. Pour cela, le processus de conception du protocole de sécurité passe par plusieurs phases. Les différentes phases de conception du protocole de sécurité sont présentées comme suit :

Phase 1 : identifier les objectifs, étudier l'état de l'art et définir les besoins et les exigences.

Cette première phase consiste à identifier les objectifs à atteindre avec ce nouveau protocole. Pour cela, il faut commencer par faire une étude de l'état de l'art des solutions de sécurité existantes. Cette étude porte sur les avantages et les inconvénients de chaque solution ainsi que les lacunes présentes dans chaque solution. Il faut relever aussi les failles et les vulnérabilités de ces solutions. Il s'ensuit alors la description des besoins et des exigences à couvrir par ce nouveau protocole. La présentation de cette phase fait l'objet des chapitres 2 et 3.

Phase 2 : spécifier l'architecture de sécurité et le protocole SVSP

L'architecture de sécurité sur laquelle se base le protocole SVSP est décrite en détail dans le chapitre 4. Le principe de fonctionnement du protocole SVSP est présenté ainsi que les éléments et les différents services de sécurité de base dans le chapitre 5. Les niveaux de sécurité sont déterminés et définis dans cette même sous phase. Les différents services de sécurité et leurs options sont examinés. Les différents types d'échange de ce protocole sont bien spécifiés et détaillés. Des scénarios de ce protocole sont élaborés pour assurer la mise en œuvre de ce protocole dans des cas réels.

Phase 3 : Analyser et intégrer SVSP dans des architectures de téléphonie

Une fois que le protocole SVSP est bien défini, une nouvelle phase d'étude est nécessaire. Cette phase permet de déterminer les possibilités, les limitations et les astuces d'intégration de ce protocole dans les architectures de téléphonie existante. Cette étude qui fait l'objet du chapitre 6, doit assurer une garantie de suivre les objectifs du protocole SVSP dont la principale est de garder un haut niveau d'abstraction vis-à-vis de l'infrastructure du réseau.

Phase 4 : analyser et mettre en œuvre SVSP

Cette phase consiste à analyser l'architecture et les services de sécurité offerts par le protocole SVSP vis-à-vis d'autres protocoles de sécurité similaires. Cette analyse montre aussi sa complexité et son niveau d'extensibilité.

Suite à cette analyse, une mise en œuvre de SVSP permet d'effectuer des tests pour évaluer le protocole. Cette évaluation permet de garantir que les fonctionnalités décrites dans les spécifications du protocole répondent au cahier de charge de SVSP.

Il s'agit aussi de soumettre ce protocole à des tests de vulnérabilités et d'évaluer sa robustesse face à certains types d'attaques. Dans ce stade, nous choisissons le langage de programmation et la possibilité de réutiliser une API de téléphonie existante ou de la modifier pour implémenter SVSP. Cette phase sera décrite dans le chapitre 7 de ce mémoire.

L'exploitation de ce protocole amène à prendre des décisions concernant la satisfaction de toutes les exigences de sécurité qui ont été attribuées. Le résultat de cette évaluation peut servir à affiner ce protocole ou bien à faire des modifications qui peuvent atteindre le stade de la spécification générale du protocole.

4.4 Principales caractéristiques du protocole SVSP

Le protocole SVSP est un protocole de sécurisation de la voix. L'objectif est de présenter une indépendance vis a vis de l'infrastructure du réseau sous-jacent. SVSP est une charpente pour effectuer des communications de téléphonie sécurisées de bout en bout. Il offre les services de sécurité nécessaires pour la téléphonie. Ainsi SVSP assure l'authentification de l'usager, la confidentialité et l'intégrité des messages échangés, le non-rejeu des messages et la non répudiation de l'appel. SVSP est convenable pour la protection de la téléphonie dans des environnements hétérogènes (interconnexion de réseaux fixes et sans fils). Ce protocole décrit un échange de messages sécurisés entre deux entités ou plus, a savoir entre deux usagers pour une communication point a point ou pour une communication entre les usagers a travers une tierce entité de confiance. Il peut interagir avec le réseau sous-jacent pour utiliser les services de sécurité du réseau sous-jacent.

Les principales caractéristiques de SVSP sont les suivantes :

1. *Définition de deux méthodes d'authentification*

   Sous la même considération de sécurité présentée ultérieurement, SVSP propose une authentification de l'usager suivant deux méthodes différentes. La première méthode consiste en une authentification via une entité tierce de confiance. Cette authentification est basée sur un identifiant unique protégé et sauvegardé par l'usager et l'entité de confiance ainsi qu'une clé pré-partagée entre les deux entités pour sécuriser l'échange. Ceci permet d'authentifier les usagers sans avoir besoin de recourir aux lourdes opérations cryptographiques à base de certificats effectuées dans la deuxième méthode. La deuxième méthode repose sur l'échange des certificats des deux utilisateurs, de leurs identités sécurisées par la clé public de l'utilisateur distant. Ceci permet aussi de négocier les paramètres de sécurité qui seront utilisés pour assurer la confidentialité et l'intégrité des échanges.

2. *Support de mécanisme de non-répudiation*

   Un des mécanismes de sécurité nécessaires à la téléphonie est la non-répudiation de l'appel. Le passage à travers une entité tierce de confiance pour l'authentification des usagers permet de contrôler l'appel établi. Ce contrôle se fait par la fourniture d'un Token à l'usager juste après son authentification auprès de cette entité de confiance et après l'authentification de l'usager distant. Une fois l'appel sécurisé est terminé, l'usager doit rendre le Token à l'entité de confiance. Ce Token servira à l'élaboration d'un rapport sur l'appel signé par l'entité de confiance ou par l'usager lui-même.

3. *la mobilité des usagers*

Puisque le protocole SVSP est de niveau applicatif, il permet ainsi d'être portable sur tout type de réseau. SVSP repose sur l'utilisation d'un identifiant unique et universel. Ainsi, l'usager, pour être authentifié n'a pas besoin d'utiliser l'identification offerte par le réseau auquel il appartient mais a un identifiant fourni par une entité tierce de confiance et qui est universelle. De ces deux caractéristiques nous pouvons tirer un avantage pour assurer la mobilité des usagers sécurisés par SVSP. La technologie des cartes à puce est en réelle ébullition. De nouvelles améliorations y sont apportées. Pour cela, une carte à puce peut être utilisée pour porter ce protocole et les paramètres de sécurité associée et ainsi assurée la mobilité des usagers.

4. *les exigences du protocole SVSP*

Lorsqu'on parle de la sécurité des données, on évoque généralement les protocoles de sécurité et les exigences ou services qu'assurent ces protocoles (tels que l'authentification, la confidentialité, l'intégrité, le non-rejeu et la non-répudiation). Dans ce contexte, l'aspect de définition des exigences apparaît important, notamment la définition des capacités et des limites du protocole SVSP, mais aussi quelques services optionnels que ce protocole peut satisfaire.

Les principaux besoins et exigences pour la conception de SVSP sont les suivants:

- Négociation simple des services de SVSP.
- Intégration des fonctionnalités des protocoles de sécurisation des réseaux sous-jacents
- Protection d'identité des usagers vis-à-vis des personnes non autorisées.
- Support de plusieurs méthodes d'authentification.
- Négociation des paramètres de sécurité pour chaque service.
- Centralisation des politiques de sécurité.
- Traçage des communications sécurisées établies.
- Utilisation des algorithmes de chiffrement robustes et fiables.

4.5 Le protocole SVSP

Dans ce paragraphe, nous définissons l'architecture de notre protocole SVSP.

4.5.1 Architecture du protocole

Dans le cas de notre architecture SVSP, nous avons décidé d'utiliser une carte à puce pour héberger l'application SVSP ainsi que les paramètres de sécurité (e.g., l'identifiant de l'usager et la clé pré-partagée) relatif à la politique de sécurité. L'utilisation d'une puce nous permet de stocker d'une façon sécurisée les données sensibles de l'utilisateur.

Le stockage de la clé secrète partagée ou des certificats ainsi que l'identifiant unique peut se réaliser durant la phase de la personnalisation de la carte.

Cela consiste à créer un fichier de base servant à stocker les paramètres de sécurité sensibles (e.g., la clé partagée, la clé privée et l'identifiant de l'usager) dans une page mémoire inaccessible aux attaques.

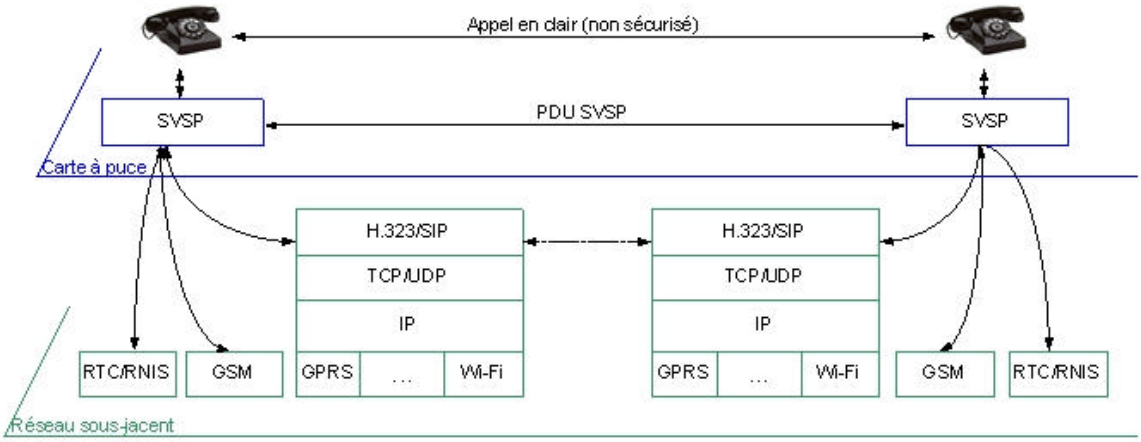

Figure 4.1 - L'architecture du protocole SVSP

Le protocole SVSP est formé de trois sous-couches :

a. La sous-couche application SVSP traite des services de sécurité offerts par les protocoles et qui est formé par les modules suivants :
   - module SVSP qui offre les services de sécurité, c'est l'application SVSP qui définit les différentes phases de l'établissement d'une session SVSP ainsi que le choix des services de sécurité à appliquer
   - module de chiffrement ; c'est dans ce module que s'effectuent toutes les opérations de chiffrement symétrique et asymétrique
   - module certificats de signature et de chiffrement ; son rôle est de gérer les certificats des TAAs et des abonnés (s'il y a lieu)
   - module de génération et de gestion de clés; ce module est utilisé pour la gestion des clés de chiffrement et d'intégrité
   - module de vocodeur ; il est utilisé sur une plate-forme où l'implémentation n'a pas accès au codec de voix.
b. La sous-couche message SVSP qui modélise le format du fragment SVSP et qui sera traité dans la section 5.10.2
c. La sous-couche API SVSP qui gère l'interconnexion entre le protocole SVSP et le réseau téléphonique sous-jacent.

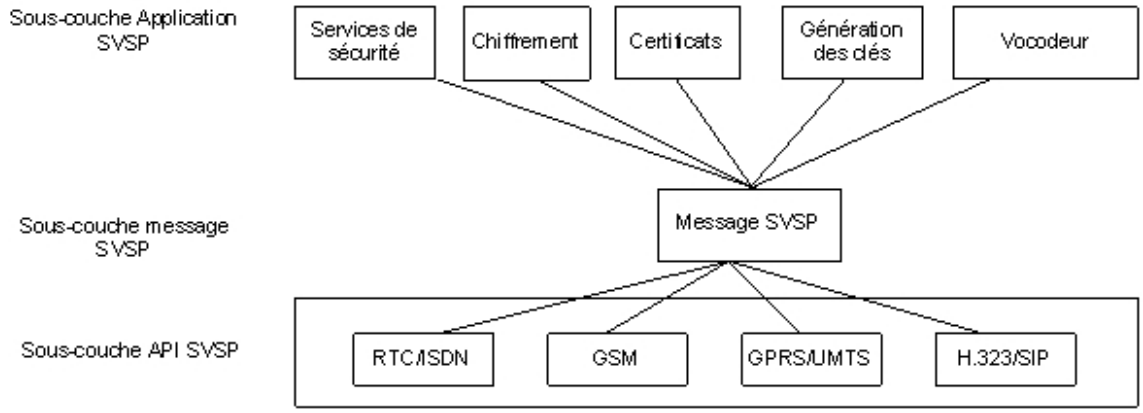

Figure 4.2 - Les sous-couches du protocole SVSP

La relation entre les deux premières sous-couches et les modules de chaque sous-couche est gouvernée par le choix de la politique de sécurité requise par l'usager et déployée dans le réseau téléphonique. Le module vocodeur est utilisé dans les réseaux de téléphonie qui ne permette pas l'accès à son codeur de voix.

4.5.2 Format du message SVSP

Les commandes SVSP échangées entre les usagers et les TAA au niveau applicative devrait être reconstruit en des messages de niveau transport avec des champs d'entête et de données utiles. La figure 4.3 montre le format du message SVSP.

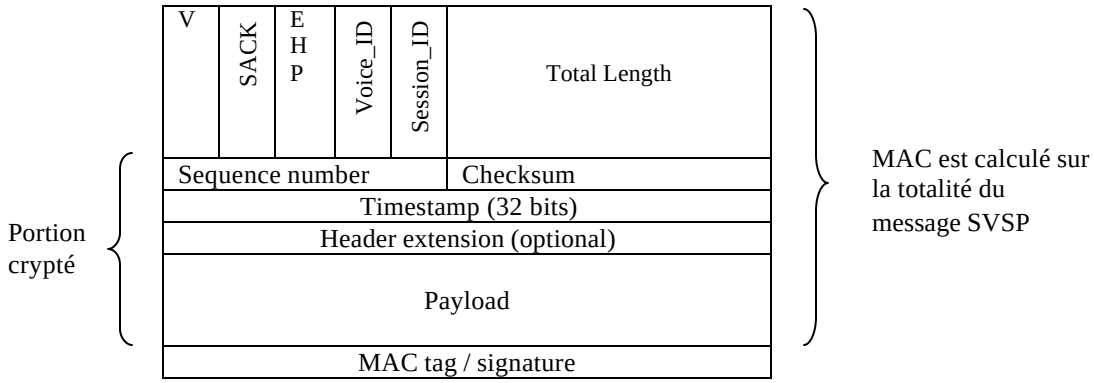

Figure 4.3 - Le format du message SVSP.

- V (2 bits): indique la version du protocole SVSP (version 1)
- SACK (1 bit): ce bit, s'il est positionné à 1, indique l'utilisation de l'acquittement sélectif entre les deux usagers distants.
- EHP (Extension Next Pointer) (1 bit): indique la présence d'une entête d'extension dans le message SVSP
- Total Length (16 bits): c'est la longueur totale incluant les entêtes et le champ données utiles du message SVSP
- Voice_ID (4 bits): indique le type de protocole de voix utilisé durant la communication par l'usager d'origine qui envoie le message SVSP.
- Session_ID (8 bits): identifie la session SVSP
- Sequence Number (16 bits): le numéro de sequence est incrémenté de 1 pour chaque message SVSP envoyé. Ce numéro de séquence sera utilisé avec une technique de fenêtre glissante pour éviter les attaques par rejeu des paquets de voix. Il permet aussi de déterminer les pertes de paquets SVSP, de maintenir une liste des de non-rejeu perdus et de récupérer le séquencement des paquets. La valeur initiale du numéro de séquence devrait être aléatoire, non prédictible pour rendre les attaques par connaissance connues sous le non de «known-plaintext attacks on encryption» plus difficile.
- Checksum (16 bits): une fonction polynomiale calculée sur le paquet SVSP.
- Timestamp (32 bits): utilisé pour le service de non-répudiation.
- Header extensions: les entêtes d'extension sont utilisées pour reporter des erreurs, négocier des paramètres de sécurité entre les usagers ou pour transporter des

informations spécifiques entre les différentes entités comme la taille du segment (PMTU) sur le chemin emprunté par la communication SVSP.

- Payload: les données utiles en provenance des codecs audio ou vidéo ou des données de signalisation.
- MAC tag (32bits): condensât calculé sur tous les champs du fragment SVSP en utilisant une fonction de hachage (e.g., HMAC [25]). Le condensât est ajouté au message SVSP dans les échanges entre les usagers et les TAA d'une part puis entre les usagers de bout en bout. Il sera remplacé par une signature pour les messages échangés entre les TAAs.

4.5.3 Les entêtes d'extension

L'objectif de l'entête d'extension est de transporter des informations spécifiques entre les usagers d'une part et entre l'usager et la TAA d'autre part. La figure 4.4 montre le format générale d'une entête d'extension. L'information transportée dans l'entête d'extension peut être de trois types: messages de commandes, messages d'erreur, messages de contrôle et messages informationnels. Un message SVSP peut transporter jusqu'à 4 entêtes d'extension.

Type	Code	F	Réservé	longueur
Données utiles				

Figure 4.4 - format de l'entête d'extension

- Type (4 bits): indique le type de l'entête d'extension
- Code (8 bits): indique le sous-type du type de message transporté par une entête d'extension
- F (1 bit) : un flag indique s'il est positionné à 1, ceci indique qu'une autre entête d'extension suit cette entête. S'il est positionné à 0, ce bit indique que c'est le dernier message.
- Réservé (3 bits): réservé pour une utilisation future
- Longueur (16 bits): indique la longueur de l'entête d'extension
- Donnée utile: contient les données utiles transportées par l'entête d'extension.

Parmi les messages qui peuvent être transporté dans une entête d'extension, nous citons:

- Les messages de commandes et de contrôle: *OpenChannel*, *CloseChannel*, *SetCipherSpec*, etc.
- Les messages d'erreurs : message de demande de retransmission durant la phase d'authentification des usagers. En cas de perte de messages déterminée par la liste de rejeu maintenu par chaque usager, une entité envoie un message de retransmission qui acquitte implicitement les messages bien reçus et demande la retransmission des messages perdus. La technique utilisée est basée sur les acquittements sélectifs (SACK). Un temporisateur est déclenché la phase d'initialisation (authentification) pour palier aux pertes dues à un réseau non fiable. Si le réseau par lequel passe le protocole SVSP est fiable, ce temporisateur peut ne pas être utilisé. Ce choix est gardé pour l'implémentation.

- Les messages informationnels : ces messages transportent les adresses source et destination des entités communicantes au besoin.

4.5.4 Les messages d'extension d'entête

a. Adaptation du message de transport

Chaque message d'authentification de SVSP doit contenir dans un seul datagramme afin d'éviter la fragmentation dans les réseaux IP et donc la nécessité de réassemblage au niveau du récepteur. Ce qui va augmenter encore plus les délais. Pour éviter la fragmentation de IP, les implémentations de SVSP doivent déterminer la taille du MTU et envoyer des messages plus petits que le MTU. Les implémentations de SVSP doivent fournir une façon pour les applications pour déterminer la valeur du PMTU (ou alternativement la taille maximale de datagramme de l'application, qui est le PMTU moins l'entête du message SVSP). Si l'application tente d'envoyer un message plus grand que le MTU l'implémentation de SVSP doit produire une erreur, ainsi évitant l'envoi d'un paquet qui sera fragmenté.

Certains protocoles de transport, tel que DCCP [DCCPdraft] fournissent leurs propres numéros de séquence. Quand SVSP est transporté par-dessus ces transports, le SVSP et les nombres de séquence de transport seront présents. Bien que ceci introduit une petite inefficacité, la couche de transport et les numéros de séquence de SVSP servent à des buts différents et donc pour la simplicité conceptuelle c'est supérieur pour utiliser les deux nombres de séquence. A l'avenir, les extensions de SVSP peuvent spécifier si ceci le permet l'usage seulement d'une seule série de numéros de séquence pour le déploiement de SVSP dans des environnements contraignants. Certains protocoles de transport, tel que DCCP, fournissent un contrôle de congestion pour le trafic transportée par-dessus eux. Si la fenêtre de congestion est suffisamment étroite, ceci nécessiterait peut être des retransmissions des messages SVSP ainsi que l'expirations des temporisateurs. Quand SVSP est utilisé par-dessus de tels transports, il doit être pris en compte pour ne pas remplir la fenêtre de congestion.

b. Découverte du PMTU

En général, la philosophie de SVSP est d'éviter de traiter les problèmes de PMTU. La stratégie générale sera de commencer avec un MTU relatif à la capacité du réseau dans lequel SVSP est déployé et s'ensuit alors une mise à jour si les événements l'exigent. La taille du MTU peut être déterminée à partir de l'identifiant Voice_ID spécifié dans l'entête du message SVSP. Ainsi, cet identifiant peut nous indiquer le type de réseau (e.g., RNIS) et nous pourrons alors tirer la taille du MTU transporter par ce réseau.

Le PMTU devrait être initialisé de l'interface qui sera utilisé pour envoyer des paquets. Si SVSP est déployé dans un réseau IP, il peut reposer sur le PMTU de IPv6 [RFC1981] ou bien utiliser les procédures de découverte de MTU de IPv4. Dans un réseau IP, la taille du MTU doit s'adapter à l'état du réseau.

c. Expiration du Temporisateur et Retransmission

SVSP utilise une pause et retransmission simples suivant la machine d'état présentées à la figure 4.5. Les usagers de SVSP envoient le premier message

(U_RUCA.Req). Dès qu'ils envoient ce premier message, ils passent à l'état INITIALISATION.

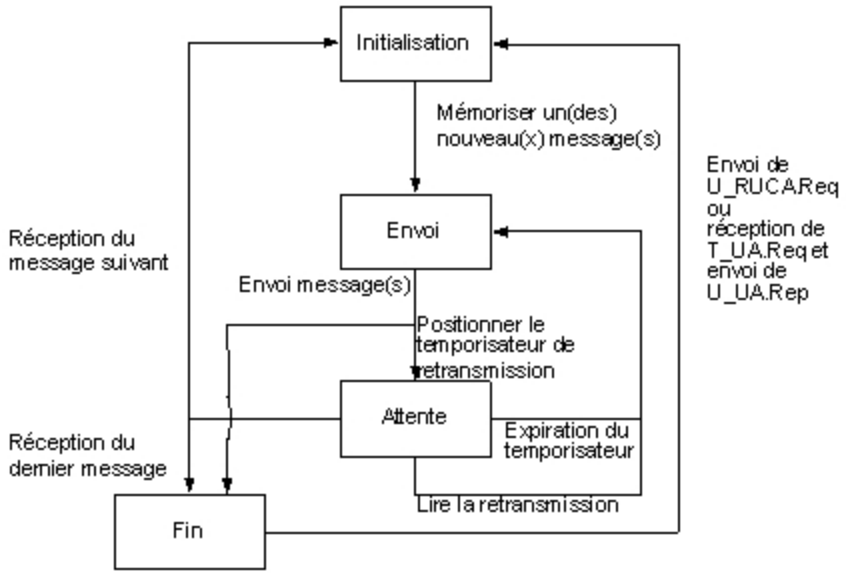

Figure 4.5 – machine à état de l'expiration des temporisateurs et de la retransmission

La machine à état a trois états fondamentaux :

- Dans l'état d'INITIALISATION, l'implémentation prépare le terrain pour l'envoi d'un nouveau message (ou un groupe de message). Ils mémorisent alors dans les mémoires tampons le(s) message(s) à envoyer après avoir vider le tampon et passe à l'état ENVOI.

- Dans l'état ENVOI, l'implémentation transmet les messages en mémoire. Une fois les messages ont été envoyés, l'implémentation entre alors dans l'état FIN si ceci est le dernier message SVSP à envoyer, ou, si l'implémentation compte recevoir plus de messages, positionne un temporisateur de retransmission et entre alors dans l'état ATTENTE.

Il y a trois façons pour sortir de l'état ATTENTE :

- Le temporisateur de retransmission expire : l'implémentation passe à l'état ENVOI, et retransmet le message, initialise le temporisateur de retransmission et retourne à l'état ATTENTE.
- L'implémentation lit un message retransmis de l'entité distante : elle passe alors à l'état ENVOI, où elle retransmet le message, initialise le temporisateur de retransmission, et retourne à l'état ATTENTE. Dans ce cas, la réception d'un message double indique une expiration du temporisateur chez l'entité distante et suggère donc que le message précédent a été perdu.

- L'implémentation reçoit le message prochain : si ceci est le dernier message, l'implémentation passe à l'état FIN. Si l'implémentation a besoin d'envoyer un nouveau message, il passe à l'état INITIALISATION.

d. Valeur des temporisateurs

Bien que le choix des valeurs des temporisateurs soit le choix de l'implémentation, un mauvais choix peut mener à des problèmes de retransmission rapide si le temporisateur expire rapidement surtout sur des liens congestionnés.

Les implémentations doivent utiliser une valeur de temporisateur qui soit adéquate pour chaque type de réseau téléphonique. Un exemple peut être cité sur un réseau IP où le temporisateur doit être initialisé à 1 seconde (le minimum défini dans RFC 2988 [RFC2988]) et doublé la valeur à chaque retransmission, jusqu'à pas moins que 60 secondes [RFC 2988].

Les implémentations doivent retenir la valeur du temporisateur actuel jusqu'à ce qu'une transmission sans perte arrive, à ce moment la valeur du temporisateur peut être remise à la valeur initiale. Après une longue période d'oisiveté, non moins que 10 fois la valeur actuelle du temporisateur, les implémentations peuvent initialiser le temporisateur à la valeur initiale.

e. SetCipherSpec

Le message de SetCipherSpec est un message qui indique les paramètres de sécurité (e.g., AES, HMAC-SHA1, etc.) choisis par les deux entités communicantes pour sécuriser la communication vocale.

4.6 Les services de sécurités offerts par SVSP

4.6.1 Authentification

L'un des plus importants services de sécurité considérés est de s'assurer de l'identité et de l'authenticité de l'usager avec qui nous communiquons. Avant de débuter une communication téléphonique, nous avons bien besoin d'authentifier notre correspondant. Le protocole SVSP supporte différentes méthodes d'authentification de l'usager en se basant sur l'architecture du réseau de téléphonie et celle de la sécurité appliquée à ce réseau. Dans le cadre de cette étude, SVSP offre l'authentification indirecte de l'usager en passant par une tierce partie de confiance ou bien l'authentification directe de l'usager lui-même. Les deux formes d'authentification requièrent l'échange de messages spécifiques SVSP et le déroulement de différents mécanismes de sécurité.

Dans l'authentification de l'usager via une tierce partie de confiance, l'usager considère cette entité tierce comme une entité de confiance et partage avec elle un secret. L'échange de message nécessaire à l'authentification sera basé sur des manipulations de ce secret partagé. Les entités tierces utiliseront des mécanismes basés sur des certificats pour sécuriser les échanges entre elles tout en déroulant le protocole SVSP.

Dans l'authentification directe de l'usager, chaque usager possèdera un certificat X.509. L'authentification mutuelle des usagers nécessite l'échange de messages SVSP sécurisés par des mécanismes basés sur les certificats.

Un troisième cas se présente entre un usager direct et un usager se trouvant derrière une entité tierce. Cette authentification sera une combinaison des deux cas précédents.

4.6.1.1 L'usager - TAA

Chaque usager possède un identifiant unique et universel obtenu de la TAA racine et livré par la TAA de rattachement. Cet identifiant est créé à partir des informations obtenues depuis la carte d'identité nationale (ou passeport) de l'usager. L'usager débute une session SVSP par l'envoi d'une requête de demande d'authentification à son TAA. La TAA va vérifier si cet utilisateur est inscrit dans sa base de donnée UDB.
Si l'usager est déjà un abonné de la TAA, cette dernière va lui envoyer un nombre aléatoire. Suite à la réception de cette nombre, l'usager va l'utiliser avec son identifiant unique comme pour générer un condensât. Le calcul du condensât (HRES) est effectué par une fonction de hachage appliquée au couple (nombre aléatoire, identifiant unique). Ce nombre aléatoire servira aussi pour générer une clé de session en utilisant la clé maîtresse partagée et l'algorithme de chiffrement spécifié dans le message **UserCapabilitySet**. Le condensât est envoyé par l'usager à la TAA. Elle le comparera avec celui qu'elle a calculé en se basant sur les informations prises sur l'usager de sa base de donnée UDB. Suite à la comparaison, si la TAA obtient le même résultat, elle émet un message **OpenChannel** vers l'usager. Ce message indique à l'usager qu'il peut commencer une session SVSP avec l'usager distant. Le message **OpenChannel** contient aussi un **Token** fourni par la TAA. Le **Token** reçu, la clé pré partagée et l'algorithme de génération de clé (i.e., Diffie-Hellman or RSA) seront utilisés pour dériver les clés de session. Ces clés dérivées seront utiles pour le chiffrement et la formation d'un MAC à ajouter aux messages SVSP échangés entre les usagers communicants.

Si l'abonné n'est pas abonné à la TAA (cas de mobilité), deux scénarios se présentent :

1- la TAA consulte la GTAA pour retrouver la TAA à laquelle l'usager est abonné. Pour cela, elle envoie le champ Voice_ID présent dans l'entête du message SVSP et l'adresse source de l'entête d'extension à la GTAA pour déterminer vers quelle réseau ou domaine du réseau il faut faire les recherches. Si GTAA retrouve la TAA de l'usager, elle émet un message à la TAA hôte pour lui donner les paramètres d'accès à la TAA d'origine. La TAA hôte entre en communications avec la TAA d'origine pour obtenir les informations nécessaires sur l'usager ou bien elle jouera le rôle de relais de messages entre la TAA d'origine et l'usager. Dans ce cas là, Elle sera transparente et n'affectera pas les messages SVSP échangés entre l'usager et la TAA d'origine.
2- la TAA rejette la requête d'authentification et envoie à l'usager un message **AuthenticationReject**.

4.6.1.2 TAA –TAA / usager-usager

Pour l'authentification des TAAs entre eux, suivre le même scénario d'authentification entre deux entités communicantes présenté au paragraphe 4.8.1. Cependant, pour éviter de refaire le même processus à chaque nouvel appel entre deux TAAs, cette

authentification entre ces deux TAAs sera maintenue par des messages *keepalive*. La fréquence des messages *keepalive* s'espace dans le temps si le nombre d'appels SVSP devient moins fréquent entre les usagers de ces deux TAAs. Si l'espacement entre deux messages *keepalive* consécutifs atteint une certaine valeur, le processus d'authentification sera répété lors d'une nouvelle session entre leurs abonnés respectifs.
L'inconvénient majeur de cette méthode d'authentification est le délai introduit par le premier appel SVSP entre deux TAAs non authentifiés qui sera plus élevé que pour les appels suivants.

4.6.2 Confidentialité

Un service de confidentialité est offert aux communications de téléphonie sécurisées par le protocole SVSP. Il permet de négocier les paramètres de sécurité nécessaires au chiffrement des données échangées. Ces paramètres englobent et ne se limitent pas à l'algorithme de chiffrement qui sera utilisé, le protocole de génération des clés de session, etc. Parmi les paramètres qui peuvent être négociés la possibilité d'utiliser les services de sécurité offerts par l'infrastructure du réseau sous-jacent. Ainsi les usagers peuvent déterminés le type de réseau de l'usager distant. Si le réseau de téléphonie de l'usager distant permet d'utiliser un protocole de sécurité commun entre les deux usagers, les deux usagers négocient les paramètres relatifs à ce protocole de sécurité qui offre le service de confidentialité.

Un exemple est l'utilisation du protocole SRTP pour la téléphonie sur IP. L'utilisation de SRTP permet d'éviter les redondances dans les services de sécurité. SRTP est mieux qualifié pour sécuriser le trafic RTP. Il permet de minimiser les *overhead* protocolaires et peut être accompagné par un protocole de compression d'entête, le protocole CRTP.

Pour sécuriser les données échangées, SVSP offre un large choix d'algorithme cryptographique essentiellement l'algorithme de chiffrement par block AES [24].

4.6.3 Intégrité

Les mécanismes d'intégrité sont déployés pour la protection contre l'altération des données par des tierces parties malveillantes. Cette intégrité peut être assurée soit en utilisant la signature électronique ou les fonctions de hachage (e.g., HMAC-SHA1 or HMAC-MD5). Pour éviter la lourdeur des opérations introduites par la signature électronique, cette signature sera essentiellement utilisée par une des méthodes d'authentification décrite par le protocole SVSP et pour la génération du rapport de non-répudiation. Lors des échanges des données entre les entités communicantes, l'intégrité des messages sera assurée par les fonctions de hachage qui sont moins lourdes à manipuler. Chaque message SVSP transportera un condensât (*Message Authentication Code* (MAC) tag) qui sera calculé sur tout le message SVSP. Ce condensât sera remplacé par une signature électronique au besoin.

4.6.4 Non-rejeu

Pour assurer la protection contre les attaques par rejeu, chaque message SVSP est représenté par un numéro de séquence, par un identifiant de la session et par un champ d'horodatage. Ces triplets constituent les points d'entrée sous forme de vecteur dans une liste de rejeu maintenue par chaque récepteur. Une fenêtre glissante sera appliquée à

chaque vecteur en vérifiant dans l'ordre suivant l'identifiant de la session, le numéro de séquence puis l'horodatage. Seuls les paquets relatifs a cette session seront vérifiés et les autres seront rejetés. La technique de la fenêtre glissante possède 3 bornes ; une borne inférieure (borne de gauche) des numéros de séquence en dessous de laquelle tous les messages seront rejetés. Tous les messages SVSP qui arrivent et non encore reçus entre les deux bornes d'extrémité seront acceptés. Tous les messages qui arrivent après de la borne supérieure seront mémorisées pour être traités. La taille de la mémoire doit être choisie en fonction de la QoS requise. A mesure que la fenêtre glisse vers la droite, la liste de rejeu sera mise à jour.

Le SN peut être géré de la même manière suivante que celle d'IPSec ESP. Ainsi, le premier message d'un flux entre deux correspondants SVSP, possède un SN égal à 1. A chaque nouveau message émis par une pile SVSP, le SN est incrémenté de 1. A la réception d'un message, SVSP gère une fenêtre glissante (de 16, 32 ou 64 positions par exemple) permettant de préciser les numéros de séquence déjà reçus dans cette fenêtre. L'algorithme de détection du rejeu est le suivant :

- Un message avec un SN plus petit que la borne inférieure de la fenêtre est détruit;
- Un message avec un SN déjà validé dans la fenêtre est un rejeu et doit être traité en conséquence (alerte à l'usager distant ou au TAA, journalisation,…);
- Un message avec un SN dans la fenêtre mais non encore validé est accepté;
- Un message avec un SN dépassant la borne supérieure de la fenêtre provoque le glissement de celle-ci vers ce nouveau SN.

Le fait d'ignorer les messages dont le SN est en deçà de la fenêtre peut entraîner des réémissions de messages gérées par TCP si nous sommes dans un réseau IP (mais pas par UDP, pour lequel les messages sont définitivement perdus). Toutefois, la liste de rejeu peut suivre le modèle décrit dans [RFC2401].

4.6.5 Non-repudiation

En téléphonie, pour s'assurer que l'usager a bien effectué un appel avec tel correspondant à telle date, nous avons recours aux factures émises par l'opérateur. Les usagers font confiance aux opérateurs émetteurs de ces factures. Pour rendre ces documents authentiques, il suffit que ces rapports soient émis par l'usager lui-même ou bien par une entité tierce de confiance, signé par l'entité émettrice du dudit rapport. Le rapport peut prendre le format présenté à la figure 4.6.

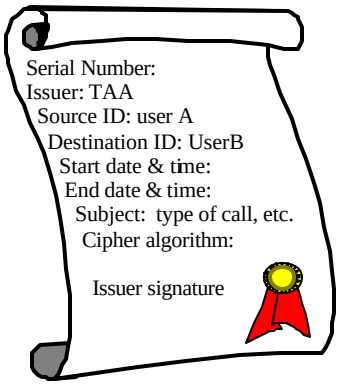

Figure 4.6 - Non-repudiation report

Quand l'usager veut terminer une session de téléphonie sécurisée par SVSP, il envoie les messages **U_ESMS.req** et **U_ESMS.rep** contenant les commandes **EndSecureMediaSession** et **EndSecureMediaSessionAck**. Ensuite les usagers envoient respectivement a leur TAA les messages **U_ESMS.ind** contenant la commande **closeChannel** et le **Token**. A la réception de ces informations, la TAA génère un rapport

de non-répudiation, le chiffre avec la clé partagée avec son usager et le signe avec son certificat de signature. Le **Token** servira au TAA pour la génération du rapport. La TAA garde une copie du rapport dans sa base de donnée RDB. Le rapport peut être utilisé pour des objectifs de facturation de la session.

4.7 Considération de gestion des clés

Pour la génération des clés de session, différentes méthodes peuvent être utilisées, soit en se basant sur des algorithmes asymétriques ou symétriques. Nous pouvons citons a titre d'exemple les algorithmes RSA [RSA], Diffie-Hellman [Dif76], ISAKMP [RFC2408], MIKEY [RFC 3830], etc. Les algorithmes comme RSA demandent beaucoup de ressources et nécessitent un traitement lent. Pour cela, afin d'éviter de surcharger l'API SVSP des usagers, nous recommandons d'utiliser le protocole MIKEY, crée à la base pour les sessions multimédias dans des environnements de réseaux fixes et sans fils.

4.8 La sécurisation des communications entre les entités

Le protocole SVSP repose sur l'architecture de sécurité présentée dans le chapitre précédent. SVSP gère les communications entre les autorités de confiance TAA et les usagers, entre les usagers et entre les GTAA et définit les mécanismes pour la sécurisation des échanges entre ses entités. Chaque TAA gère un certains nombre d'usagers abonnés et leur délivre un identifiant unique lors de leur inscription. L'usager considère son TAA comme une entité de confiance avec laquelle il communique en sécurisant l'échange par l'utilisation d'une clé pré-partagée. L'identifiant de l'usager ainsi que d'autres paramètres de sécurité du protocole SVSP son fournis a l'usager sur une carte à puce ou bien par un petit programmable (patch) à récupérer d'un serveur. Cette carte peut contenir aussi a titre d'exemple un profile d'utilisateur (identité, carnet d'adresse book, sa restriction d'appel, etc.) et une politique de sécurité.

La carte à puce est une clé qui permet au terminal de fonctionner en mode sécurisé. Si elle n'est pas insérée, le terminal fonctionne en mode non sécurisé. Elle peut être protégée par un mot de passé (typiquement 4 chiffres). Elle peut contenir l'API de protocole SVSP pour éviter de surcharger le terminal.

Puisque l'usager considère son TAA comme une entité de confiance, un échange sécurisé basé sur une PKI n'est pas nécessaire entre l'usager et la TAA. La solution par des clés pré-partagées réduit les opérations cryptographiques fournies par le terminal de l'usager, la surcharge du nombre de messages échangés entre l'usager et la TAA. Ainsi, la communication entre l'usager et la TAA est chiffrée à l'aide de la clé pré partagé, avec une fonction de hachage pour assurer l'intégrité des messages. La TAA chiffre et signe ses messages échangés avec la TAA distante en utilisant ses certificats X.509.

La politique de sécurité est basée sur le choix des besoins et exigences en terme de sécurité comme définie dans le chapitre 3 paragraphe 9.

4.8.1 Un scénario d'appel de base SVSP entre deux entités

Pour établir une communication de voix sécurisée par SVSP, ceci nécessite l'ouverture préalable d'un canal de communication en mode natif entre les deux usagers. Suite à cette ouverture du canal, l'usager a le choix de basculer d'un mode en clair en un mode SVSP sécurisé avec l'usager distant. Durant la phase d'authentification mutuelle des usagers, ils utilisent des certificats de chiffrement et de signature. Ces certificats seront utiles aux usagers pour sécuriser l'échange des identifiants uniques des utilisateurs. Suite à la phase d'authentification, les échanges de messages entre les deux usagers sont sécurisés par un chiffrement symétrique basé sur une clé pré-partagée entre ces deux entités. L'intégrité des messages est assurée par une fonction de hachage à sens unique. Cet identifiant sera utilisé pour la génération des paramètres de sécurité tel les clés de session et les clés pour les fonction de hachage.

Le déroulement du protocole SVSP (fig.4.7) débute par l'envoi de messages d'initialisation **U_RUCA.req** (1, 2) en mode transparent, non chiffré. Ces deux messages comportent le **UserAuthenticationRequest** pour demander une session d'authentification avec l'usager distant**,** le **UserCapabilitySet**, ainsi que le certificat de l'usager et un nombre aléatoire généré par ce dernier. Puisque ce nombre sera utilisé ultérieurement pour la génération des condensâts, des clés de sessions, etc., il faut bien le protéger. Dans ce cas là, seul ce nombre sera chiffré avec la clé public du récepteur du message. L'usager récepteur validera en premier lieu le certificat reçu avec l'entité de certification de confiance (CA). Si le certificat est valide, il procède alors au chiffrement du nombre aléatoire avec la clé public de l'usager distant et à la poursuite de la session SVSP. L'usager récepteur garde dans une base de donnée les certificats reçus avec une correspondance avec le détenteur du certificat. Deux cas se présentent :

- Si l'usager connaît son interlocuteur distant et possède son certificat valide dans sa base de données, alors le message **U_RUCA.req** (1, 2) sera échangé avec le nombre aléatoire chiffré avec la clé public du récepteur du message.
- Si l'usager ne possède pas le certificat de son interlocuteur distant ou possède un certificat périmé, il procède alors à l'envoi d'un message **U_RUCA.req** (1, 2) exempte du « random ». Dès qu'il reçoit le certificat de son interlocuteur et qu'il le valide avec l'entité de certification alors il pourra envoyé dans un second message **U_RUCA.req** le « random » chiffré avec la clé publique de l'interlocuteur.

Le **UserCapabilitySet** transporte les modes opératoires et autres paramètres de sécurité de l'usager émetteur (e.g., usager A) de ce message. Les modes opératoires permettent aux deux entités de choisir les paramètres de sécurité commune (e.g., choix de l'algorithme de chiffrement, la fonction de hachage, etc.). Les modes opératoires seront utiles pour débuter une session de media de bout en bout sécurisée par SVSP.

A la réception des valeurs aléatoires, les usagers calculs des condensâts avec leur identifiant unique et la variable aléatoire reçue. Les condensâts résultants *(e.g.,* $HRES=hash(N_{d,B}, ID_A)$ seront transportés dans les messages SVSP **U_RUCA.rep** (3, 4). Ces messages transportent un acquittement sur le choix des modes opératoires respectifs et une clé pré-partagée K_f. Cette clé est obtenue par le chiffrement de l'identifiant de

l'usager initialiseur du messages avec la clé publique de l'usager distant *(e.g., $f(KBpub, ID_A) = K_{f,A}$)*.

L'usager distant (UserB) va tirer l'identifiant de l'usager d'origine (UserA) en utilisant sa clé privée. Puis il va appliquer une fonction de hachage sur l'identifiant tiré et la variable aléatoire qu'il a généré et envoyé à l'usager d'origine (UserA). Si les deux résultats sont équivalents, alors l'usager distant, toujours selon la fig.4.7, envoie dans le message **U_RUCA.conf** (5, 6) un message **UserAuthenticationConfirm** et un message **OpenChannel** vers l'usager opposé qui contient un **Token**. Ce même message transporte aussi un message **Cverify**. Le **Cverify** inclut un condensât de tous les messages envoyés/reçus depuis le message d'initialisation **U_RUCA.req.** Ce condensât est ensuite chiffré avec la clé privée de l'usager (UserB) qu'il envoie vers l'usager (UserA). Ce dernier vérifie le certificat de l'usager (UserB) ainsi que sa signature appliquée sur le condensât des messages **U_RUCA.req** et **U_RUCA.rep.** À ce moment, les deux usagers échangent les messages **SetCipherSpec**. Les deux usagers font appel au protocole **SetCipherSpec** afin de déclencher le chiffrement des échanges selon les choix effectués dans les phases précédentes. Il est à noter que le **Token** servira pour la génération de la preuve de non-répudiation à la fin de l'appel sécurisé par SVSP.

Les messages **U_ESMS.req** (7) et **U_ESMS.rep** (8) terminent une session SVSP entre les usagers. Ces messages seront suivis par les messages **U_ESMS.ind** (9, 10) qui délivre à l'usager opposé le *Token*. A la réception du *Token* livré avec le message **CloseChannel**, chaque usager génère le rapport de non-répudiation et l'envoie à l'usager opposé dans un **U_ESMS.conf** (11, 12) chiffré par la clé public de l'usager distant et signé l'usager d'origine.

Il est à noter que tous les messages de la session d'appel téléphonique seront cryptés avec la clé K_f de chaque usager et hachés avec une fonction HMAC pour éviter les opérations par clé public/clé privée qui demandent beaucoup de ressource. Tous les messages SVSP possèdent un numéro de séquence et un horodatage.

Pour la génération des clés de sessions, différentes méthodes peuvent être utilisées, soit en se basant sur des algorithmes asymétriques ou symétriques. Nous pouvons citons a titre d'exemple les algorithmes RSA, DH, ISAKMP, MIKEY, etc. Les algorithmes comme RSA demandent beaucoup de ressources et nécessitent un traitement lent. Pour cela, afin d'éviter de surcharger l'API SVSP des usagers, nous recommandons d'utiliser le protocole MIKEY, crée à la base pour les sessions multimédias dans des environnements de réseaux fixes et sans fils.

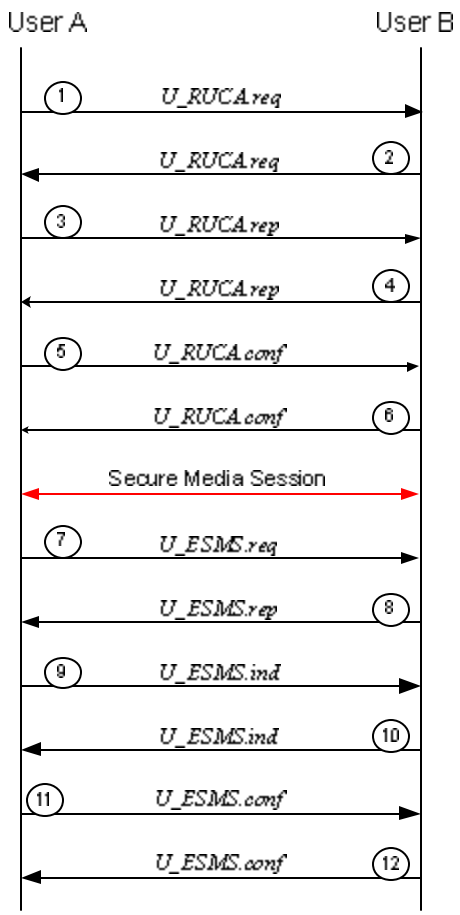

Figure 4.7 - Scénario d'un appel basic entre 2 entités

Légende des messages SVSP:
U_RUCA.req: User_RemoteUserCapabilityAuthentication.request
U_RUCA.rep: User_RemoteUserCapabilityAuthentication.reply
U_RUCA.conf: User_RemoteUserCapabilityAuthentication.confirmation
U_ESMS.req: User:EndSecureMediaSession.request
U_ESMS.rep: User:EndSecureMediaSession.reply
U_ESMS.ind: User:EndSecureMediaSession.indication
U_ESMS.conf: User:EndSecureMediaSession.confirmation

Le tableau 3 montre tous les messages SVSP échangés entre les deux entités communicantes, leurs significations, et les paramètres qu'ils transportent.

Table 3 - Tableau des messages SVSP du scénario d'appel entre deux entités

No.	Nom	UA→UA	UA←UA	Paramètres
1	U_RUCA.req:	Obligatoire	-	UserAuthenticationRequest, Cuser, UserCapabilitySet, Rand=$N_{d,A}$ = ($N_{i,A}$ || date)
2	U_RUCA.req :	-	Obligatoire	UserAuthenticationRequest, Cuser, UserCapabilitySet, Rand=$N_{d,B}$ = ($N_{i,B}$ || date)
3	U_RUCA.rep	Obligatoire	-	UserCapabilitySetAck, f(KBpub, ID_A)= $K_{f,A}$ HRES=hash($N_{d,B}$, ID_A)
4	U_RUCA.rep	-	Obligatoire	UserCapabilitySetAck, f(KApub, ID_B)= $K_{f,B}$ HRES=hash($N_{d,A}$, ID_B)
5	U_RUCA.conf	Obligatoire	-	UserAuthenticationConfirm, openChannel(Token_B), session_ID, Cverify, SetCipherSpec
6	U_RUCA.conf	-	Obligatoire	UserAuthenticationConfirm, openChannel(Token_A), session_ID, Cverify, SetCipherSpec
7	U_ESMS.req	Obligatoire	-	EndSecureMediaSession
8	U_ESMS.rep	-	-	EndSecureMediaSessionAck
9	U_ESMS.ind	Obligatoire	-	CloseChannel, Token = (Nd,{A,B} || Session_ID || date), Session_ID
10	T_ESMS.conf	Obligatoire	optionnel	MessageReport, Session_ID

4.8.2 Un scénario d'appel de base SVSP entre quatre entités

Pour établir une communication de voix sécurisée par SVSP, ceci nécessite l'ouverture préalable d'un canal de communication en mode natif entre les deux usagers comme le montre la figure 4.8. Suite à cette ouverture du canal, l'usager a le choix de basculer d'un mode transparent en un mode SVSP sécurisé avec l'usager distant. Les échanges de messages entre l'usager et le TAA sont sécurisés par un chiffrement symétrique basé sur une clé pré-partagée entre ces deux entités. L'intégrité des messages est assurée par une fonction de hachage à sens unique. De même, la sécurisation des échanges entre les TAAs est assurée par le déploiement de certificats de chiffrement et de signature pour assurer leur aspect confidentiel et leur intégrité. Le déroulement du protocole SVSP débute par l'envoi de messages d'initialisation (1, 2) chiffré avec la clé pré-partagée entre l'usager et la TAA. Dans ce cas-là, tout le message SVSP sera chiffré. Ces messages comportent deux autres messages SVSP **UserAuthenticationRequest,** et **UserCapabilitySet**. A la réception du message contenant le message **UserAuthenticationRequest,** les TAAs envoient des messages (2 bis, 2 ter) aux usagers pour leur livrer un nombre pseudo aléatoire. Ce nombre sera utilisé pour la génération des

clés de sessions pour chiffrer le trafic direct entre les usagers et le TAA. Il sera utilisé pour générer les condensâts nécessaires à l'authentification des usagers par les TAAs.

Le message contenant le **UserAuthenticationRequest** est une requête de demande d'authentification envoyée par l'usager vers son TAA. Elle est destinée aussi au TAA et usager distant pour commencer une session SVSP.

Le message contenant le **UserCapabilitySet** transporte les modes opératoires et autres paramètres de sécurité de l'usager émetteur (e.g., usager A) de ce message. La TAA utilisera les modes opératoires choisis par l'usager (e.g., algorithme de chiffrement, fonction de hachage) pour pourvoir sécuriser les échanges de messages vers l'usager. Le rôle du TAA est de livrer ensuite ce message vers l'usager destinataire (e.g., usager B). Les modes opératoires sont formés par les algorithmes cryptographiques, les fonctions de hachage, et autres paramètres de sécurités à être négociés entre les usagers. Les modes opératoires seront utiles pour débuter une session de media de bout en bout sécurisée par SVSP. A la réception des valeurs aléatoires, les usagers calculs des condensâts. Les condensâts résultants seront transportés dans les messages SVSP (3, 3bis) vers les TAAs respectifs. Les TAAs feront le même calcul et compareront le résultat obtenu avec le résultat reçu. Si les deux résultats sont équivalents, alors la TAA distante, toujours selon la fig.4.8, envoie un message SVSP contenant le champ **UserAuthenticationConfirm** (4, 4bis) en direction du TAA d'origine suivi d'un message SVSP contenant le champ **OpenChannel** (4bis) vers l'usager qui lui est connecté (e.g., usager B). Une fois que le TAA d'origine reçoit la confirmation d'authentification de l'usager distant, il émet un message SVSP contenant un champ **OpenChannel** (4 ter) à son propre usager.
Avec ce dernier message, l'authentification des usagers est terminée. Il s'ensuit alors une phase d'acquittement des modes opératoires respectifs des usagers par l'échange de messages SVSP contenant le champ **UserCapabilitySetAck** (5, 6). Les messages SVSP (5, 6) font appel à l'algorithme **UserSetCipherSpec** pour dérouler les paramètres de sécurité du mode opératoire choisi par les usagers et qui sont nécessaires pour la sécurisation du trafic téléphonique.

Dans le **T_UA.Req** (2 ter), la TAA envoie son certificat *CTAA* qui sera chiffré par la clé de session générée à partir de la clé PSK et le nombre aléatoire envoyé par le TAA. L'usager compare le certificat reçu avec le certificat qu'il possède. Si c'est le même il envoie au TAA un message SVSP **U_UA.Rep** (3) et **U_RUCA.Rep** (3 bis) contenant la commande **CTAA_Verify** pour lui indiquer qu'il a bien vérifier la compatibilité du certificat. Si c'est différent, dans ce cas là l'usager doit mettre à jour chez lui le certificat du TAA (certificat expiré). Ici il faut faire attention a ce qu'un man-in-the-middle ne fait le travail, donc dans le message **CTAA-Verify**, il faut que l'usager envoie un nombre aléatoire concaténé avec l'identifiant de la TAA chiffré avec la clé pré-partagée, le serveur doit tirer le nombre et l'envoyer vers l'usager dans un message SVSP **T_RUCA.rep** (4 bis, 4 ter) contenant la commande **CTAA_Verify_Ack** pour que l'usager puisse authentifier la TAA. Ce message est chiffré avec la clé de session dérivée de la clé PSK. Il transporte aussi le résultat d'une fonction PRF effectué sur tous les messages émis par le serveur vers l'usager. L'usager effectue la même fonction PRF sur tous les messages reçus de la TAA et met à jour sur sa carte à puce le certificat du TAA.

Note : La commande **CTAA_Verify_Ack** est optionnelle et nécessaire uniquement si l'usager reçoit un certificat différent de celui sauvegardé sur sa carte.

Les messages SVSP **U_ESMS.req** (7, 8) contenant les commandes **EndSecureMediaSession** et **EndSecureMediaSessionAck** terminent une session SVSP entre les usagers. Ces messages seront suivis par une commande **CloseChannel** transportée dans un message SVSP **U_ESMS.ind** (9) issu de l'usager vers son TAA. A la réception du Token livré par le message **U_ESMS.ind** (9), le TAA génère le rapport de non-répudiation et envoie **messageReport** à l'usager dans un message **T_ESMS.conf** (10) chiffré par la clé pré-partagée et signé par le TAA.

Pour la génération des clés de sessions, différentes méthodes peuvent être utilisées, soit en se basant sur des algorithmes asymétriques ou symétriques. Nous pouvons citons a titre d'exemple les algorithmes RSA, DH, ISAKMP, MIKEY, etc. Les algorithmes comme RSA demandent beaucoup de ressources et nécessitent un traitement lent. Pour cela, afin d'éviter de surcharger l'API SVSP des usagers, nous recommandons d'utiliser le protocole MIKEY, crée à la base pour les sessions multimédias dans des environnements de réseaux fixes et sans fils.

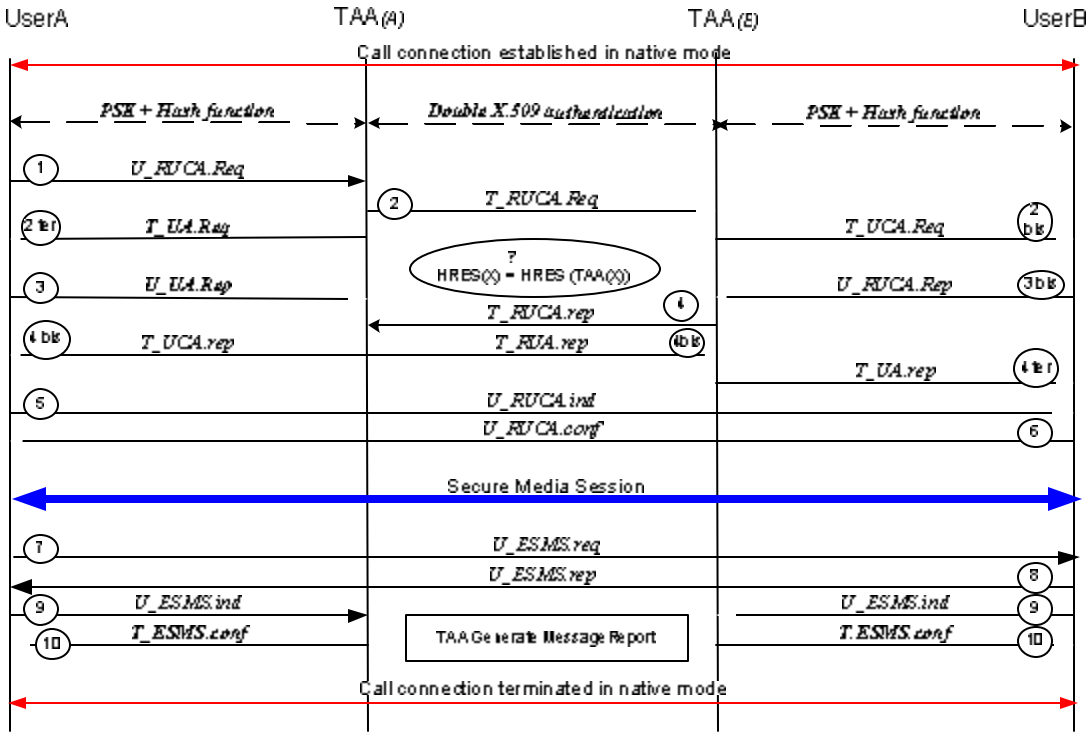

Table 4 - Tableau des messages SVSP du scénario d'appel entre quatre entités

No.	Nom	UA↔TAA	TAA↔TAA	TAA↔UA	UA↔UA	Paramètres
1	U_RUCA.Req	Obligatoire	-	-	-	UserAuthenticationRequest, UserCapabilitySet
2	T_RUCA.Req	-	Obligatoire	-	-	UserAuthenticationRequest, UserCapabilitySet
2 bis	T_UCA.Req	-	-	obligatoire	-	UserAuthenticationRequest, UserCapabilitySet, Nd,B = (Ni,B || date), TAA_Certificate
2 ter	T_UA.Req	-	-	Obligatoire	-	C_TAA, Nd,A = (Ni,A || date),
3	U_UA.Rep	Obligatoire	-	-	-	HRES = hash (Nd,A, IDA), f(Kpub, Rand)=CTAA-verify
3 bis	U_RUCA.Rep	-	-	Obligatoire	-	HRES = hash (Nd,A, IDA), f(Kpub, Rand)= CTAA-verify, UserCapabilitySet
4	T_RUCA.rep	-	Obligatoire	-	-	UserAuthenticationConfirm, UserCapabilitySet
4 bis	T_RUA.rep	-	optionnel	-	-	UserAuthenticationConfirm
4 bis	T_UCA.rep	Obligatoire	-	-	-	OpenChannel, Token = (Nd,A || Session_ID || date), Session_ID, UserCapabilitySet, CTAA_verify_Ack = Rand
4 ter	T_UA.rep	-	-	Obligatoire	-	OpenChannel, Token = (Nd,A || Session_ID || date), Session_ID, CTAA_verify_Ack=Rand
5	U_RUCA.ind	optionnel	Optionnel	Optionnel	Obligatoire	UserCapabilitySetAck, UserSetCipherSpec
6	U_RUCA.conf	Optionnel	Optionnel	Optionnel	Obligatoire	UserCapabilitySetAck, UserSetCipherSpec
7	U_ESMS.req	-	-	-	Obligatoire	EndSecureMediaSession
8	U_ESMS.rep	-	-	-	Obligatoire	EndSecureMediaSessionAck
9	U_ESMS.ind	Obligatoire	-	Obligatoire	-	CloseChannel, Token = (Nd,{A,B} || Session_ID || date), Session_ID
10	T_ESMS.conf	Obligatoire	optionnel	Obligatoire	-	MessageReport, Session_ID

4.9 Les politiques de sécurité associées

Les paramètres de sécurité définissent le comportement relatif à la sécurité du système. En utilisant des politiques de sécurité, les administrateurs peuvent appliquer, de manière centralisée ou distribuée, les niveaux de sécurité requise pour protéger des systèmes de téléphonie.

Lorsque nous déterminons les paramètres d'une politique de sécurité, le caractère organisationnel et fonctionnel de cette politique doit être pris en compte. Par exemple, les

niveaux de sécurité nécessaires pour une unité d'organisation contenant un PABX et des abonnés seront très différents de ceux d'un haut fonctionnaire d'état.

La définition des politiques de sécurité et la gestion du contrôle d'accès sont toujours un problème dans les protocoles de sécurité. Nous définissons ci-dessous quelques politiques de sécurité associés au protocole SVSP :

- Politique de sécurité libre

Cette politique ne nécessite aucun niveau de sécurité et donc l'appel téléphonique passe en clair entre les entités communicantes.

- Politique de sécurité sûre de l'infrastructure

Cette politique requiert l'authentification de l'infrastructure du réseau vis-à-vis de l'usager. Donc elle nécessite une authentification du TAA.

- Politique de sécurité sûr du distant

Cette politique est similaire à la précédente mais l'entité à authentifier peut être aussi bien l'usager distant ou la TAA auquel l'usager d'origine est rattaché.

- Politique de sécurité sûr du distant avec chiffrement

Cette politique contribue à authentifier l'entité distante avec possibilité de négocier certains services de sécurité avec le réseau sous-jacent (e.g., possibilité d'utiliser SRTP dans un contexte VoIP pour sécuriser le flux média)

- Politique de sécurité sûr du distant avec chiffrement de bout en bout

Cette politique impose l'utilisation de tous les services de sécurité offerts par SVSP.

4.10 Considérations de sécurité

- Utiliser une méthode pour la génération des nombres pseudo aléatoires qui ne demandent pas beaucoup de ressources et d'opérations de calcul lors de la négociation des paramètres de sécurité, celui qui possède le minimum commun des paramètres de sécurité impose la configuration des paramètres communs pour la sécurisation des médias. Le message *SetCipherSpec* instancie l'algorithme de spécification des paramètres de sécurités pour basculer dans une phase de chiffrement symétrique et d'intégrité par fonction de hachage.
- La reprise d'une session établie : Une nouvelle session sécurisée peut être établie sur la base d'une session sécurisée existante. Ce qui nous permet de ne plus ré exécuter la négociation complète ainsi que ses calculs cryptographiques.
- SVSP identifie la session par le paramètre *session_id*. Lorsque l'usager veut reprendre une session, il envoie le message U_RUCA.req indiquant l'identificateur de la session sécurisée à reprendre. La TAA de son côté vérifie s'il existe une correspondance dans son cache. Deux cas possibles sont à envisager :

  • S'il trouve la correspondance et s'il accepte de rétablir une connexion sécurisée pour la session indiquée, la TAA enverra un message *T_UCA.rep* qui portera l'identificateur de cette session trouvée en cache. A ce moment, l'usager répond par le message *SetCipherSpec*.

  • Si l'identificateur de la session n'est pas trouvé en cache, le serveur crée un nouvel identificateur: l'usager et la TAA exécutent une négociation d'authentification complète.

4.11 Conclusion

Dans ce chapitre nous avons présenté le protocole SVSP dont l'objectif principal est de sécuriser les communications téléphoniques dans un environnement d'interconnexion de différents réseaux de téléphonie. Le but de SVSP est d'être indépendant de l'infrastructure du réseau sous-jacent. SVSP offre les services de sécurité nécessaires à la sécurisation de la téléphonie. Il présente deux méthodes d'authentification et une fonction de non-répudiation de l'appel qui est quasi-inexistante dans les solutions de sécurité existantes actuellement. Notre première perspective est d'étudier l'intégration de SVSP dans certains réseaux de téléphonie et de déterminer les limitations qui en découlent. Notre deuxième perspective et d'analyser SVSP et de l'implémenter dans un réseau voip basé sur le protocole SIP.

Chapitre 5

5 Intégration de SVSP dans la téléphonie

5.1 Résumé

Dans ce chapitre, nous validons l'intégration de SVSP avec certaines architectures de téléphonie en étudiant l'étendue des possibilités et les limitations propres à chaque type de réseau de téléphonie. Nous proposons d'intégrer SVSP dans le réseau téléphonique traditionnel, dans le réseau mobile GSM et dans le réseau IP pour une solution de téléphonie et de voix sur IP avec le standard H.323.

5.2 Intégration dans le réseau téléphonique commuté

5.2.1 Introduction

La téléphonie est le service par excellence offert par les réseaux publics depuis une centaine d'années. Une présentation générale de l'architecture de la téléphonie et son évolution dans le réseau fixe incluant le réseau analogique RTC, le réseau numérique RNIS et le réseau de signalisation sémaphore a été faite au chapitre 2. Dans cette section, nous essaierons de proposer une solution pour intégrer le protocole de sécurité SVSP dans cette architecture, et de définir qu'elles sont les limitations et les avantages de cette intégration.

5.2.2 Présentation d'un appel type dans le réseau RTC

Décrivons brièvement l'établissement d'un appel [Rig98]. Soit un abonné relié à un central désigné par A qui appelle un correspondant. Le central détecte le décroché et envoie une tonalité d'invitation à numéroter. L'abonné compose le numéro du demandé qui est enregistré par le central A. les chiffres sont analysés pour déterminer si le numéro est complet et le central A recherche la destination de l'appel. Les premiers chiffres indiquent le numéro du central appelé. Le central A peut déterminer, par ses tables de routage, vers quel central acheminer l'appel qui peut être le central du poste demandé ou bien un central de transit (central B). Le central A va réserver un circuit libre dans le faisceau le reliant au central B et l'utilise pour transmettre le numéro du demandé. Le central B peut alors répéter la même opération vers un autre central C. la connexion va être établie de proche en proche jusqu'au central du demandé.

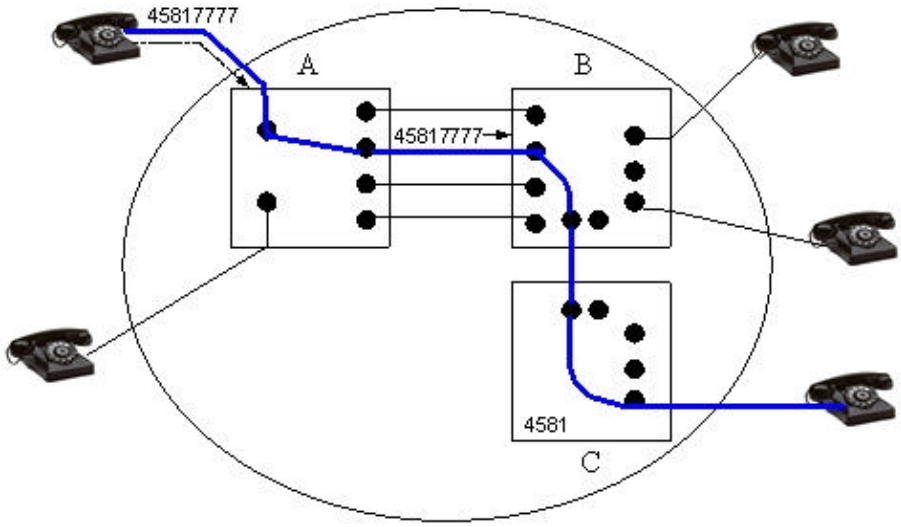

Figure 5.1 - Acheminement d'un appel en signalisation voie par voie

L'établissement de l'appel nécessite ici l'établissement d'un circuit jusqu'au demandé car celui-ci est utilisé pour le transfert de signalisation, c'est le principe de la signalisation « voie par voie ». Si à l'arrivée, il apparaît que le poste demandé est occupé, les circuits entre le demandeur et le demandé ont été réservés sans nécessité.
La signalisation est constituée des protocoles de demande d'établissement ou de relâchement d'appels. Elle peut être séparée en deux parties

- La signalisation UNI (signalisation d'abonné) : la signalisation a d'abord lieu entre l'usager et le commutateur.
- La signalisation NNI (signalisation intercommutateur) : c'est la signalisation entre les commutateurs si l'usager appelé est raccordé à un commutateurs distant. Ainsi, la fonction de traitement d'appel du commutateur de l'abonné demandeur se porte appelante vis-à-vis de la fonction de traitement d'appel du commutateur de l'usager demandé comme le montre la fig.5.2

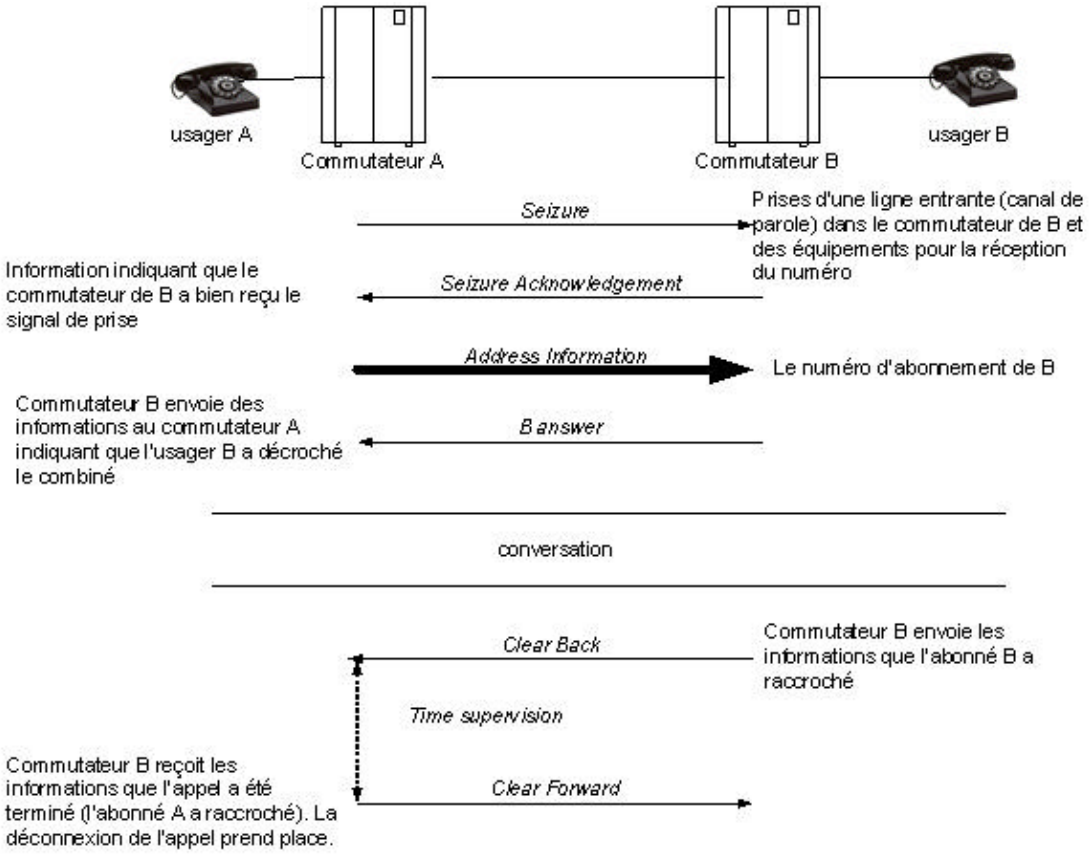

Figure 5.2 – la signalisation d'établissement d'appel entre deux commutateurs (NNI)

Dans chacune de ces deux signalisations (UNI et NNI) on peut également distinguer deux types de signaux :

- Les « signaux de lignes » : ce sont les signaux de prises ou de relâchement, soit du commutateur, soit de l'usager.
- Les « signaux d'enregistreurs » : c'est la transmission des signaux de numérotation.

Pour être correctement appelé une signalisation associée circuit, il doit y avoir une association permanente avec le canal transportant l'appel. La parole et la signalisation sont acheminées d'une façon pareille par le réseau. Certaines des variations peuvent arriver comme suit :

- La signalisation est effectuée sur le même canal que la parole (*e.g., signalisation DC, in-band*)
- La signalisation s'effectue sur le même canal de la parole mais avec une autre gamme de fréquence (*e.g., out-of-band*);
- La signalisation est transportée dans le canal 16 d'un lien E1, alors que la parole utilise d'autres canaux fixes (*e.g, signalisation PCM*)

La signalisation associée circuit (CAS) [Rig98] peut être utilisée avec tous les types de commutateurs téléphoniques.

5.2.3 Intégration avec le réseau RTC

Dans le cas d'un appel qui se présente entre deux usagers du réseau RTC, un échange de message de signalisation se fait pour l'établissement de l'appel, d'une part entre l'abonné et le commutateur auquel il est raccordé et entre les commutateurs distants.

Deux cas de figure se présentent pour intégrer SVSP dans un appel prenant place dans le réseau RTC :

1. Soit l'appel s'établit entre les deux usagers en mode natif puis à la décision des deux usagers, l'appel bascule en mode sécurisé par SVSP. On se base sur l'appel établi puis on déroule SVSP
2. Soit l'appel est établi en mode sécurisé SVSP; ici, nous pouvons tomber sur deux cas pour établir l'appel sécurisé :
    a. Soit SVSP est intégré au réseau sémaphore (signalisation SS7) utilisé dans le réseau téléphonique actuellement;
    b. Soit SVSP initie l'appel.

Essayons d'analyser chaque cas de figure et de tirer ses limitations:

Dans le premier modèle, l'appel est établi entre les deux usagers en mode natif. Les deux usagers peuvent débuter leur conversation puis à un moment donné, ils décident de sécuriser leur conversation et donc de basculer leur appel en mode sécurisé SVSP. Dans ce cas là, le circuit physique est déjà établi entre les deux usagers traversant les différents commutateurs mis en jeu. Afin de débuter la session SVSP, soit il faut ouvrir un canal en parallèle pour acheminer les messages de signalisation de SVSP puisqu'on est dans la phase de « conversation » de l'appel (fig.5.2). Il faut alors un mécanisme pour associer l'appel réel avec le nouveau canal utilisé pour assurer l'authentification des usagers. Soit il faut utiliser une signalisation spécifique dite « *in-band* » (.e.g, une gamme de fréquence bien spécifique) pour acheminer les messages de signalisation de SVSP une fois l'appel est établi en mode natif.

Ce modèle introduit beaucoup de difficulté pour son déploiement, car il nécessite le remaniement de l'infrastructure et du système de signalisation associée circuit du réseau de téléphonie qui devient un système assez ancien.

Dans le second modèle, je commencerais par le second point (point b) qui propose que l'application SVSP initie l'appel. L'usager choisit au préalable d'établir un appel sécurisé par SVSP. L'implémentation de SVSP permet d'initier un appel de donnée sur un canal de téléphonie. Ainsi, les messages de SVSP seront transportés par le réseau comme étant des blocs de données qui ne seront pas traités par le réseau de téléphonie. Cette implémentation ne nécessite pas d'introduire des changements dans le réseau de téléphonie et elle est applicable pour le modèle SVSP à deux entités. Cependant, pour que l'application SVSP puisse initier l'appel, le lecteur de carte à puce sera munie d'un modem intégré (fig.5.3) qui permettra d'établir une connexion de donnée. L'application SVSP intègrera un codec de voix qui prendra les échantillons de parole, les numérisera avant d'appliquer le protocole SVSP sur le paquet de donnée ainsi formé. La difficulté de cette intégration réside dans l'implémentation de l'application SVSP et son interaction avec le modem intégré.

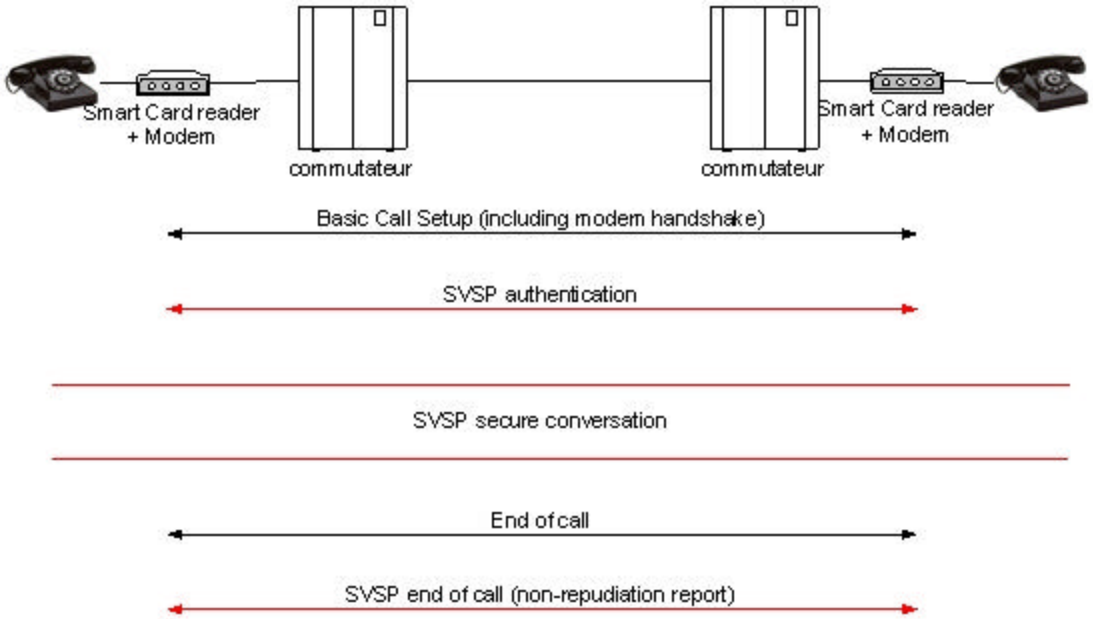

Figure 5.3 – intégration de SVSP dans un appel RTC

Il est à noter que SVSP devrait être intégré dans un petit équipement spécifique muni d'un lecteur de carte à puce et qui s'insère entre le téléphone et la ligne RTC. Cette intégration dépend aussi des limitations dues aux protocoles du réseau téléphonique commuté.

5.2.4 Intégration avec le réseau RNIS

Dans le cas d'un réseau RNIS, pour relier un terminal numérique (ETTD) il n'est pas nécessaire de disposer d'un modem puisque le réseau et le terminal sont numériques. Cependant, les débits de transmission du terminal et du réseau étant différents, il faut placer un adaptateur (appelé parfois modem RNIS par abus de langage) entre le terminal et le réseau.

Le RNIS suppose que l'usager puisse signaler en même temps qu'une communication est déjà établie, soit pour établir une autre communication en simultané (par exemple une communication de données alors qu'une communication téléphonique est déjà en cours), soit pour activer un complément de service, etc.

Une signalisation voie par voie (dans la bande) ne permet pas ce mode de fonctionnement. En effet, une fois que la conversation est démarrée, il n'y a plus de possibilité d'échanger des signaux d'enregistreurs en signalisation voie par voie car les données de signalisation se mélangeraient avec les données de conversation. Il est donc nécessaire pour signaler pendant la conversation de disposer d'un système de signalisation hors bande sur tous les arcs du chemin établi dans le réseau. Là nous nous trouvons confrontés à un problème de connexité : il faut que tous les nœuds du chemin établi disposent d'un système de signalisation sémaphore tant à l'UNI qu'au NNI.

Au NNI, la signalisation hors bande utilisée pour les appels RNIS s'appuie sur le réseau de signalisation sémaphore avec le sous-système utilisateur ISUP (pour *ISDN*

User Part). A l'UNI, la signalisation utilisée est multiplexée temporellement sur la même paire téléphonique utilisée par l'usager, ce qui lui conserve son caractère hors bande.

Si nous voulons suivre les deux cas de figure cités dans la section précédente, nous trouvons ce qui suit :

1. ce modèle consiste à établir un appel téléphonique non sécurisé sur un canal B à 64 kbps. Si les usagers décident de basculer dans un mode sécurisé SVSP, alors ceci nécessite la signalisation de ce choix entre les deux usagers sur le canal D puis l'ouverture d'un nouvel appel sécurisé par SVSP sur un nouveau canal B.
2. ce modèle nécessite l'intégration de SVSP dans le réseau sémaphore que nous verrons dans la section suivante.

Les majeures difficultés dans l'intégration de SVSP avec le réseau RNIS résident dans l'interfaçage avec la signalisation transportée sur le canal D et la création d'un lecteur de carte à puce avec une interface RNIS.

5.2.5 Intégration avec le réseau SS7

Par opposition à la signalisation voie par voie, la signalisation par canal sémaphore peut se définir comme une méthode dans laquelle une seule voie, le «canal sémaphore », achemine, grâce à des messages étiquetés, l'information de signalisation se rapportant à une multiplicité de circuits ou à des messages de gestion et de supervision. Le principal avantage de la signalisation sémaphore est de permettre l'échange de signalisation indépendamment de l'établissement d'un circuit. On parle alors de « signalisation non associée circuit ».

Pour pouvoir déployer SVSP, nous avons vu précédemment plusieurs modèles parmi lesquels figurent un modèle d'implémentation de SVSP comme étant un sous-système utilisateur du réseau sémaphore que nous nommerons SAP (pour *Security Application Part*). Ainsi, les échanges de messages SVSP s'effectueront dans le plan contrôle du réseau téléphonique, d'une façon transparente à l'usager.
L'architecture d'intégration de SVSP se fera comme le montre la fig. 5.4 suivante:

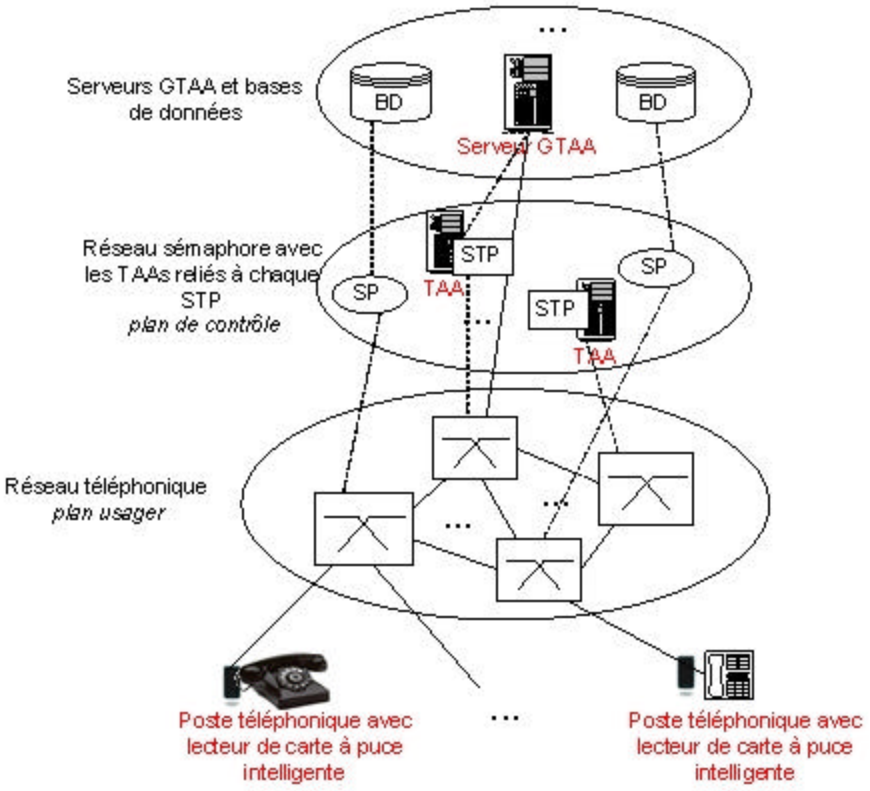

Figure 5.4 - Architecture de l'intégration de SVSP dans le réseau sémaphore

Chaque poste téléphonique est munie d'un lecteur de carte à puce intelligente. Les TAAs seront déployés au niveau des points de transferts (STP), alors que le GTAA sera connecté au SCP (pour *Signaling Control Point*).

5.2.5.1 Description générale du protocole SAP proposé

Le protocole SAP (Security Application Part), régira l'ensemble des échanges entre les équipements de l'architecture de sécurité proposée au chapitre 4 dans le réseau sémaphore. Il offrira les fonctions de signalisations nécessaires au service de sécurité d'une communication de voix par le protocole SVSP. Il aura principalement trait à toutes les fonctions de sécurité qui permettent à un usager de sécuriser son appel.

Le protocole SAP sera structuré comme un catalogue de requêtes et de réponses possibles tirées de SVSP et adaptées au réseau sémaphore. Il s'appuiera ainsi de façon naturelle sur TCAP (pour *Transaction Capabilities Application Part*) [Rig98], lui-même utilisant SCCP (pour *Signaling connection Control Part*) [Rig98], implanté au dessus de MTP (pour *Message Transfer Part*) [Rig98]. Le SCCP sera seulement utilisé en classes 0 et 1, c'est-à-dire en mode non connecté. Cette hiérarchie est représentée à la figure 5.5 avec les différents niveaux d'encapsulation.

Les messages SAP de requêtes sont associés à des composants TCAP (*Invoke*), les réponses sont associées à des réponses TCAP (*Result*).

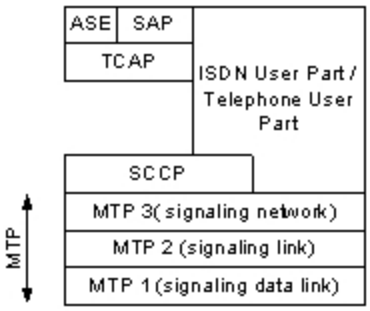

Figure 5.5 – modèle de référence du réseau SS7

5.2.6 Conclusion

Dans cette section, nous avons proposé des solutions pour intégrer SVSP au réseau téléphonique traditionnel comme nous avons montré les difficultés et les limitations de chaque solution. Nous avons étendu notre solution pour inclure l'intégration avec le réseau sémaphore, qui à lui seul, permet une meilleure intégration de SVSP avec une grande majorité de réseau de téléphonie qui utilise les services de signalisation du réseau SS7. Cette proposition d'intégration avec le réseau sémaphore est inspirée d'un travail de recherche qui a été effectué à l'université de Georges Mason.

5.3 Intégration dans le réseau GSM

5.3.1 Introduction

La téléphonie est de loin le service le plus populaire offert par les réseaux publics, y compris les réseaux fixes et les réseaux cellulaires mobiles. Après une présentation générale des réseaux mobiles GSM/GPRS/UMTS dans le chapitre 2, nous tenterons de décrire l'environnement avec lequel le réseau GSM doit s'interconnecter afin d'établir un appel vocal sécurisé de bout en bout par le protocole SVSP en utilisant le modèle à deux entités décrit au chapitre 4.

5.3.2 Présentation générale de l'appel

Prenons l'exemple d'un appel entre un utilisateur de GSM appelé Bernard et un abonné d'un réseau de téléphonie fixe, appelé Fred. Sur le terminal de Bernard, l'application SVSP est intégrée à la carte SIM du mobile. Il existe à l'heure actuelle des cartes SIM émises par la société finlandaise Setec [Setec] supportant les PKI. Sur le terminal de Fred, un lecteur de carte intelligente est positionné entre le poste téléphonique et la ligne de transmission. L'application SVSP se trouve emmagasinée sur la carte intelligente de Fred. La transmission doit assurer le transfert des signaux entre ces 2 usagers. Le chemin de transmission traverse les équipements du réseau GSM de Bernard au point d'interconnexion avec le réseau fixe, puis l'appel sera acheminé dans le réseau fixe (PSTN) vers l'usager Fred.

L'abonné du PLMN (Bernard) doit d'abord composer le numéro du correspondant demandé dans le réseau PSTN (Fred) sur son terminal puis valider l'appel par une touche spécifique. Tant que l'appel n'est pas validé, le mobile est totalement muet vis-à-vis du

réseau, la numérotation dans GSM est dite *hors-ligne* (*off-line*), car l'abonné est relié à son central de rattachement lorsqu'il numérote.

Une fois l'appel validé, la station mobile accède au réseau pour demander un canal radio sur lequel elle peut échanger de la signalisation. Elle envoie un message qui précise l'identité du mobile, le type de service demandé (appel départ) et quelques autres informations nécessaires.

Le réseau peut engager une procédure d'authentification, puis activer le chiffrement en mode natif. Ces deux services de sécurité sont assurés par le réseau GSM sur la voie radio uniquement et ne sont pas en relation avec les services de sécurité offerts par le protocole de sécurité SVSP. Le mobile transmet ensuite le numéro du correspondant désiré. Le VMSC traite l'appel comme un appel téléphonique ordinaire. Un canal radio de trafic est alloué et le mobile est commuté sur celui-ci. Lorsque le correspondant appelé décroche, la communication est établie. La durée typique entre l'accès et la première sonnerie est de six secondes. Une fois l'appel est établi en mode natif entre les deux usagers, ils peuvent basculer en un appel sécurisé via le protocole SVSP.

Commençons tout d'abord par décortiquer l'appel en mode natif entre le réseau GSM et le réseau fixe PSTN. Depuis l'usager Bernard, le premier article rencontré est le microphone de sa station mobile. Dans cette station mobile, le signal de voix analogique de Bernard est transformé dans en un ensemble d'information numérique à un taux de 13 kbit/s, qui représente le signal de l'appel vocal. D'autres procédés changent ces informations numériques en des signaux analogiques à haute fréquence qui seront transmis par la suite sur l'interface radio.

Après avoir été détecté par une antenne d'une station de base, ce signal radio est traité pour retrouver le signal numérique représentant l'appel. Il sera transporté par-dessus les câbles coaxiaux vers un transcodeur de voix. Des 13 kbit/s de données reçues, le transcodeur de voix dérive une autre représentation numérique du signal d'appel, à un taux de 64 kbit/s suivant la norme utilisée dans la transmission du réseau fixe. Ensuite, il est routé par le MSC et les divers liens et commutateurs dans le réseau fixe PSTN jusqu'à ce qu'il atteigne le commutateur local auquel est raccordé l'usager Fred. Dans le réseau PSTN, le signal analogique est reconstruit du signal numérique à un débit de 64 kbit/s. il sera par la suite transporté sur la ligne d'abonné de l'usager Fred jusqu'à ce qu'il atteigne son téléphone, où il y aura une autre transformation du signal analogique en un signal acoustique qui devrait être reconnu comme la voix de Bernard. Cette description élémentaire d'un appel de téléphonie implique plusieurs modes de transmission pour l'appel, et donc plusieurs plans de transmission.

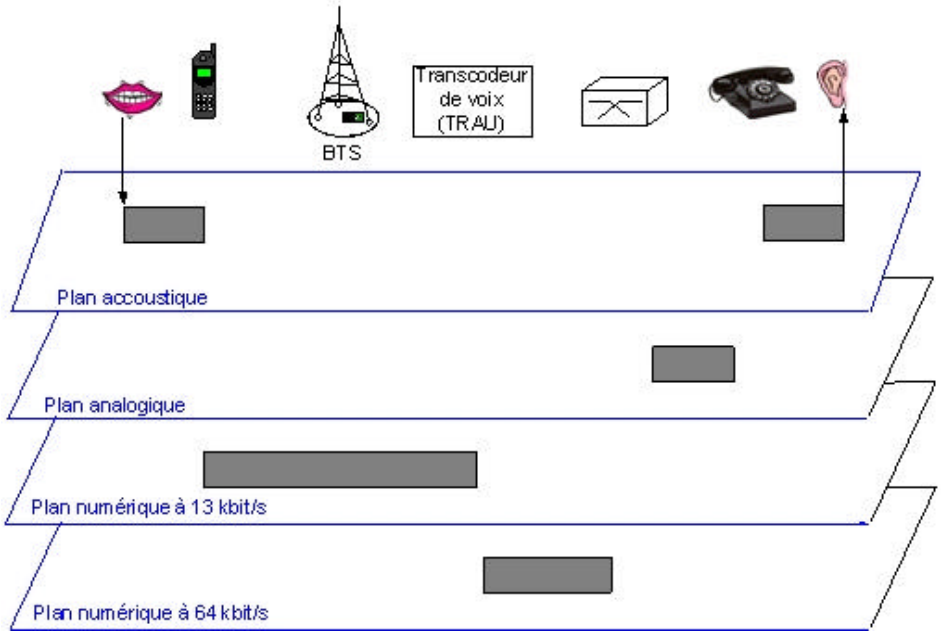

Figure 5.6 - Représentation de la voix dans les différents plans de transmission

Pour terminer la communication, lorsque l'abonné mobile raccroche, un message de déconnexion est envoyé au VMSC. Le VMSC génère un message de fin ou de libération conformément au protocole utilisé. La communication est alors terminée mais il faut libérer l'ensemble des ressources : connexions établies et ressources radio.

5.3.3 Architecture détaillée du réseau GSM

Le chapitre 2 donne l'architecture globale du réseau GSM. Comme nous avons donné un exemple d'un appel effectué entre un abonné du réseau GSM et un abonné du réseau fixe, nous allons essayer de voir plus en détail ce qui existe entre la station mobile et le IWF. Le IWF (pour *InterWokring Function*) est un ensemble de fonctions assurant les adaptations nécessaires entre le réseau GSM et les réseaux externes.
Le long du chemin de transmission, l'architecture du réseau GSM distingue entre les BTS (la Station d'Emetteur-Récepteur de Base), le BSC (le Contrôleur de Station De base) et le MSC. Entre la station mobile et les BTS est un point de référence transparent qui n'est autre que l'interface de radio sur laquelle transitent les informations sur des ondes électromagnétiques à 900 ou 1800 Mhz. L'étude des BTS/BSC/MSC est suffisante pour les aspects signalisation. Alors un nouveau équipement a été introduit : le TRAU (pour *TRanscoder/Rate Adaptor Unit*), qui est sans aucun doute un équipement de transmission, conçu pour être distinct du BSC ou du MSC. Le TRAU jouera le rôle principal dans cette section.

Le raisonnement derrière l'existence du TRAU, distinct du MSC et BSC, consiste en plusieurs points. La supposition implicite pendant la phase d'élaboration du concept appliqué au MSC était qu'il serait une réplique modifiée d'un commutateur RNIS. Comme conséquence, la transmission au niveau du MSC est très proche des

spécifications du réseau RNIS. En particulier, seulement des circuits de 64 kbit/s sont commutés. L'usage fondamental des interfaces du MSC sera de porter des circuits de 64 kbit/s compatibles aux besoins de RNIS. Le multiplexage s'effectue sur la base d'un cycle de 125 µs, chaque cycle transportant un octet par circuit.

Chaque appel entre deux abonnés GSM doit subir deux transformations (de 13 kbit/s à 64 kbit/s et, vice versa pour l'appel de voix introduisant ainsi deux opérations de transcodage entre les représentations de 13 kbit/s et 64 kbit/s), même si les deux abonnés sont connectés à la même BTS.

5.3.4 Limitations de cette intégration

La transmission numérique de l'appel par-dessus une interface de radio dans un environnement mobile est tout à fait un défi. Comme déjà mentionné, un algorithme de codage de voix numérique spécial est utilisé dans GSM, choisi pour son bas débit (13 kbit/s) et sa résistance aux taux d'erreur élevés.

Pour résumer, nous pouvons dire que la voie de transmission traversée par l'appel dans le réseau GSM peut être divisée en les segments suivants:

a. la station mobile ;
b. de la station mobile à la station de base, la voie radio ;
c. de la station de base au transcoder de voix TRAU ;
d. du transcoder de voix au MSC.

Le point de jonction séparant les segments a) à d) traité ci-dessus correspond aux endroits où une représentation de la voix est transformée à une autre représentation. Ces points de transcodage sont d'importance majeure ici, puisque la description d'un schéma de transmission est intimement reliée à la description des fonctions de transcodage correspondantes et à leur impact sur les messages du protocole SVSP. Les points de Transcodage sont le "les ascenseurs" entre les différents plans de la figure 5.6. Les points suivants de transcodage sont identifiés dans le domaine de GSM :

- transcodage d'un signal acoustique à un signal analogique, appliqué au niveau du microphone dans le terminal émetteur, et vice versa appliqué au niveau du l'haut-parleur dans le terminal récepteur; ce type de transcodage n'est pas cependant spécifique à GSM ; Si nous voulons appliquer le protocole SVSP, il faut que ce dernier apporte avec lui son propre vocodeur de voix, puisque nous n'avons pas accès au codec de voix utiliser dans le terminal.
- transcodage analogique/numérique à 13 kbit/s (et l'opération inverse), appliqué dans la station mobile. Dans ce cas là, le vocodeur de SVSP doit aussi effectuer le même transcodage de la voix analogique en numérique au même débit de 13 kbit/s afin de suivre la norme appliquée dans le réseau GSM.
- transcodage numérique/numérique de 13 kbit/s à 64 kbit/s (et l'opération inverse), appliqué dans le transcodeur de voix, dans les BTS et ou dans le TRAU. Le problème majeur de SVSP se pose au niveau de cette transformation du signal de 13 kbit/s pour être transporté sur un circuit à 64 kbit/s. en fait, il existe une étape au préalable qui consiste prendre un débit du codec à 13kbit/s (l'équivalent de 260 bits tout les 20 ms). Un codage du canal est appliqué à cet échantillon de voix

pour protéger la parole et ainsi aboutir à une trame codée, parfois appelé bloc, de 456 bits toutes les 20ms (l'équivalent d'un débit de 22,8kbit/s) puis s'ensuit son envoi sur un circuit à 64 kbit/s pour adapter le débit au réseau RNIS. Le transcodage de la voix ainsi «paquétisée » affecte énormément le signal en entier et dès lors le message SVSP sera complètement incompréhensible et irréversible au niveau du récepteur.

Suite à cette dernière remarque, on peut conclure qu'utiliser SVSP sur un circuit de voix dans GSM est irréalisable. Pour cela, nous avons pensé à deux choix pour sécuriser l'appel depuis le réseau GSM.

- soit utiliser un service de données en mode circuit (se référer au paragraphe suivant).
- soit utiliser un service de données en mode paquet. Il faut alors passer par une connexion «data » via le réseau GPRS. On se retrouve alors à établir un appel de voix sur IP et donc la sécurisation de cet appel sera traité ultérieurement dans ce chapitre (se référer à la section «Intégration dans un réseau de voix sur IP).

5.3.5 Réalisation de cette intégration via un service de données

La transmission de données en mode circuit permet de disposer sur un réseau GSM d'un service équivalent à celui disponible sur le réseau téléphonique avec un modem (ou un adaptateur dans le cas du RNIS). Le service est simple dans sa définition mais très complexe dans son implémentation. Un réseau GSM fait intervenir des transmissions sur des supports divers (voie radio ou liaison fixe). Les formats de transmission sont adaptés à chaque type de support et sont donc spécifiques à chaque support. La transmission de données fait ainsi intervenir beaucoup de traitement et de formatage, assez simples en eux-mêmes, mais dont l'enchaînement et l'imbrication forment un ensemble complexe.

La transmission de données en mode circuit nécessite l'établissement d'une communication et monopolise un canal radio différent de celui utilisé pour acheminer la voix. .

a- Eléments mis en œuvre :
La transmission de données met en œuvre dans le réseau GSM l'IWF (pour *InterWorking Function*), physiquement placé dans le MSC ou plus souvent dans un matériel placé à proximité, et le TAF, Terminal Adaptation Function, du coté terminal. Mais d'autres éléments interviennent de façon importante :

- le MSC/VLR vérifie que le service de données demandé par un terminal est bien disponible pour l'abonné et gère les ressources IWF ;
- le HLR vérifie les droits au service dans le cas d'un appel vers un mobile ;
- le BSS supporte le codage des canaux radio (comme la parole), procède à l'adaptation de débit et aux problèmes d'alignement de trame.

b- établissement d'un appel de données
L'établissement d'un appel de données comporte les mêmes étapes que l'appel vocal simple. Il comporte de plus les phases suivantes de signalisation :

- vérification de compatibilité et négociation des différents paramètres du service de données ;
- détermination du service et choix de l'IWF (InterWorking Function)
- vérification du droit au service ;
- passage éventuel du mode phonie au mode données si nécessaire.

Les données sont toujours transportées en blocs. Il est donc nécessaire que les différents équipements (MS et IWF) soient synchronisés pour détecter correctement les débuts des blocs. La phase de transfert de données peut alors commencer, soit en mode transparent T, soit en mode non transparent NT.
La définition complète du service est donnée dans un élément de service (Bearer Capapbility) présent dans le message d'établissement d'appel (SETUP). Cet élément d'information contient entre autres :

- le type de service support (mode circuit, accès à un réseau de données, etc.)
- le type de transfert sur la partie hors réseau GSM
- le mode (T ou NT)
- le débit utilisateur et le type de modem à utiliser dans l'IWF si nécessaire (V21, V22, …)
- les caractéristiques de la jonction entre l'ETTD, (pour Equipement Terminal de Traitement de Données), et le terminal GSM (transmission asynchrone/synchrone, nombre de bits de données, nombre de bits de stops, …)

Pour un appel effectué par un abonné GSM (appel sortant), la station mobile détermine le type de service soit à partir de sa configuration, soit à partir des informations transmises par l'ETTD. Elle le place dans l'élément d'information *Bearer Capability* du message SETUP. Lorsque le MSC/VLR reçoit le message, il vérifie que le service est compatible avec le profil de l'abonné et l'IWF disponible. Il détermine alors le type d'IWF et du canal de trafic TCH (pour *Traffic Channel*) à utiliser. Le MSC/VLR peut négocier certains paramètres. Le MSC/VLR alloue ensuite les ressources IWF et demande au BSC d'allouer les ressources radio nécessaires comme pour un appel normal.
Dans le cas d'un appel vers un abonné GSM, deux cas peuvent se produire : le service est précisé dans l'appel ou bien déterminé par un numéro spécifique. Le premier cas correspond à un appel GSM-GSM ou bien à un appel GSM-RNIS. Le message d'appel contient alors un élément d'information *Bearer Capability* qui spécifie le service. Le principe d'établissement est similaire au cas précédent.

Dans le cas d'un appel GSM-RNIS, nous avons vu dans la section «intégration dans le réseau PSTN » que le transfert de données entre deux ETTD reliés par le réseau RNIS nécessite une certaine adaptation de débit. Il est donc nécessaire de réaliser un certain nombre de fonctions d'adaptation appelées RA (pour *Rate Adaptation*) entre l'ETTD et le RNIS :

- RA0 transforme le flux asynchrone en flux synchrone au même débit ;
- RA1 adapte le débit utilisateur à un débit intermédiaire de 8 et 16 kbit/s
- RA2 adapte le débit intermédiaire au débit standard RNIS, à savoir 64 kbit/s.

Une fois que l'appel de donnée en mode circuit est établi entre les deux abonnés alors nous aurons un canal de donnée sur lequel peut transiter toute information et qui

court-circuite le TRAU. Donc, il n'y aura pas de transcodage du débit de 13kbit/s à 64kbit/s.

Il est fréquent d'utiliser deux MSISDN différents associés à un même IMSI pour différencier un appel de donnée d'un appel vocal. Le MSISDN est un numéro international de l'abonné mobile qui est conforme au plan E164 de l'ITU et qui est connu de l'abonner. Pour un même abonné, il aura alors deux numéros différents, un pour un appel non sécurisé SVSP et un autre pour effectuer des appels sécurisés par SVSP. L'appel entrant est pris en charge par le GMSC et parvient au HLR par un message spécifique. Le HLR détermine l'IMSI de l'abonné ainsi que le service associé au numéro. Il forme l'élément d'information *Bearer Capability* qui précise le service (par exemple circuit de données à 9600 bps en mode transparent). Cet élément est alors transmis dans un message au VMSC. L'appel se déroule ensuite de façon normale. Le VMSC détermine le type de l'IWF et indique le type de service support au mobile dans le message SETUP. Le mobile a la possibilité de négocier les paramètres du service support.

5.3.6 Conclusion

Pour assurer l'intégration de SVSP dans GSM, il faut utiliser soit un canal de donnée en mode paquet (GPRS) ou bien établir un canal de donnée en parallèle avec le canal TSH de l'appel, en mode circuit mais qui permet de dépasser le codeur de voix au niveau du TRAU. Et donc d'éviter le passage de 13kbps à 64kbps au niveau de la TRAU/MSC.

5.4 Intégration de SVSP avec H.323

5.4.1 Introduction

H.323 est maintenant le protocole dominat pour les communications voix et multimédia sur IP. La majorité des équipements de voix sur IP, des passerelles aux téléphones IP ou aux IP-PBX supportent le protocole H.323, et l'interopérabilité entre les vendeurs est maintenant excellente. De nombreux réseaux H.323 avec des équipements venant de fabricants différents sont en fonctionnement dans le monde. Dans cette section, nous essaierons de proposer une solution d'intégration de SVSP avec H.323 tout en assurant la sauvegarde de l'interopérabilité entre les équipements des différents vendeurs.

5.4.2 Présentation d'un appel entre deux postes H.323

Pour établir une conférence H.323 point à point, deux connections TCP sont requises, comme le montre la figure 5.7. D'abord, un canal Q.931 doit être ouvert avec un port bien connu de l'appelé. Les messages d'initialisation d'appel sont alors échangés comme définis dans H.225. Si l'appelé accepte l'appel, alors l'adresse IP et le port sur lesquels l'appelé doit être à l'écoute pour la connexion H.245 sont transportés à l'appelant à travers le canal Q.931.

Par la suite, l'appelant forme le canal H.245 en ouvrant une connexion TCP, utilisant pour cela l'adresse et le port indiqués. À ce moment, le canal Q.931 n'est plus exigé et peut être fermé. Le canal H.245 est alors utilisé par les deux entités pour échanger des capacités audio/vidéo et pour la détermination des maître et esclave.

Finalement, quand le transfert de données est achevé, le canal H.245 peut être utilisé pour terminer l'appel.

La figure 5.7 montre un exemple de la signalisation d'appel H.323 impliquée dans un établissement d'appel réussi où les deux points terminaux de la communication sont enregistrés au même *gatekeeper* et la signalisation d'appel directe (*direct call signaling*) est utilisée.

L'établissement d'une conférence H.323 point à point requiert principalement cinq étapes ou phases :

- phase A : initialisation de l'appel ;
- phase B : première communication et échange de capacités ;
- phase C : établissement de la communication audiovisuelle ;
- phase D : dialogue ;
- phase E : fin.

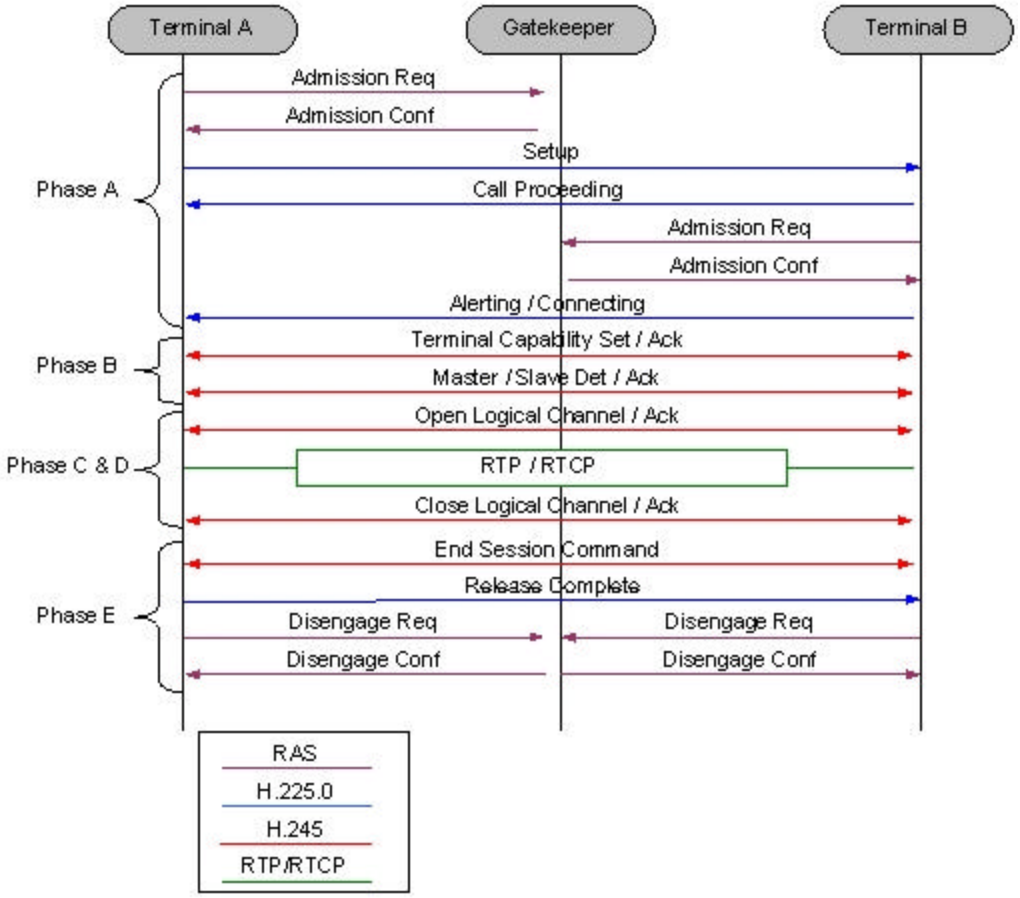

Figure 5.7 - Signalisations d'appel H.323.

5.4.2.1.1 Phase A : initialisation de l'appel

L'initialisation de l'appel a lieu en utilisant les messages de contrôle d'appel H.225 (H.225 et H.225.0 RAS). Les messages RAS sont utilisés quand un *gatekeeper* est présent dans le système, autrement seulement H.225 est utilisé.

Si le *gatekeeper* n'est pas présent, alors les deux terminaux communiquent directement. Le terminal A (le terminal appelant) envoie vers le «Canal de signalisation d'appel» (*Call Signaling Channel TSAP Identifier*) (port standard TCP 1720) du poste B un message «Initialisation» (*Setup*). Ce message comprend notamment les informations suivantes :

- taux de transfert demandé au réseau, type de codage ...
- un bloc d'information User to User avec un identificateur de protocole, l'identité H.323 de la source (une chaîne de caractères), le type de la source, l'identificateur de conférence si celle-ci est en cours et si l'on veut la rejoindre, la créer ou y inviter quelqu'un, et le type d'appel (par défaut point à point) ;

« A » peut aussi indiquer ici une adresse de canal de contrôle (H.245) où B pourra éventuellement décider de se connecter.

B répond optionnellement par un message «Traitement d'appel en cours» (*Call Proceeding*) pour indiquer que la demande d'appel a été enregistrée qui comprend éventuellement l'adresse H.245 que A devrait utiliser pour se connecter sur B, et émet encore optionnellement un message «Sonnerie» (*Alerting*) pour indiquer que l'utilisateur B est en train d'être alerté.

Enfin B termine par un message «Connexion» (*Connect*) qui comprend notamment :

- la référence de l'appel (identificateur unique de cet appel) ;
- les capacités de transfert requises selon la norme RNIS I.231 (taux de
- transfert demandé au réseau, type de codage...), informations obligatoires seulement lors d'un appel vers un *gateway* donnant accès au RTC ou au RNIS ;
- indicateur de progrès ;
- date/heure ;
- un bloc d'information User avec l'adresse de transport H.245 que A doit employer pour négocier les capacités, le Type de Destinataire, l'éventuel numéro d'identificateur de conférence.

Quand un *gatekeeper* est utilisé, il peut y avoir plusieurs configurations possibles puisque les terminaux peuvent être enregistrés au même *gatekeeper* ou à des *gatekeepers* différents, de même qu'il se peut qu'un seul terminal seulement soit enregistré à un *gatekeeper*. Il est également possible que le *gatekeeper* fonctionne soit en mode de signalisation directe (*Direct Call Signaling*) ou routée (*Routed Call Signaling*). La figure 5.7 montre un exemple d'établissement réussi d'appel où les deux terminaux sont enregistrés au même *gatekeeper* et où la signalisation d'appel utilisée est directe.

« A » initie l'échange d'*ARQ/ACF* (*Admission ReQuest / Admission ConFirm*) avec ce *gatekeeper*, qui renvoie l'adresse de transport du canal de signalisation d'appel (*Call Signaling Channel Transport Address*) de B dans l'*ACF*.

« A » utilise alors cette adresse pour envoyer le message *Setup* à B qui si tout va bien accepte l'appel et initie un échange d'*ARQ/ACF* avec le *gatekeeper*. B répond avec le message *Connect* qui contient une adresse de transport du canal de contrôle (*Control Channel Transport Address*) de H.245 que A devra utiliser dans la signalisation H.245.

Quand l'appel devrait être initialisé par l'intermédiaire d'un *gateway* ou d'un *MCU*, l'initialisation d'appel entre le *gateway* ou le *MCU* et le terminal suit une structure tout à fait semblable au scénario d'initialisation d'appel entre deux terminaux, présentée ci-dessus.

5.4.2.1.2 Phase B : première communication et échange de capacités

Une fois que les deux terminaux ont échangé les messages d'initialisation d'appel, ils doivent établir le canal de contrôle de H.245, qui sera utilisé pour l'échange de capacités (*Capabilities*) et l'établissement du canal des médias.

« A » ouvre alors un canal de contrôle H.245 vers B (ou B vers A en utilisant l'adresse du message *Setup*). Ce canal est unique pour chaque appel d'un terminal à l'autre, même si cet appel met en jeux plusieurs flux audio (langues) et/ou vidéo. Le numéro logique H.245 de ce canal est toujours 0.

Soit ce canal H.245 est ouvert par B lorsqu'il reçoit le message *Setup*, soit il est ouvert par A lors de la réception du message *Alerting* ou *Call Proceeding*. L'adresse et le port à employer ont été donnés dans l'un de ces messages.

Le premier message H.245 envoyé est *Terminal Capability Set*, qui comprend notamment les informations suivantes :

- un numéro de séquence ;
- les capacités de multiplexage de flux (*Stream Multiplex Capabilities*) ;
- table de capacités (*Capabilities Table*) : contient les possibilités d'échange et de compressions audio ou vidéo, de chiffrements et d'échanges de données : par exemple type de codages vidéo acceptés, types de codage audio, paramètres de la norme d'échange de données T.120.

Chaque terminal envoie ce message à l'autre. À la réception du message, A et B accusent réception par un message *Terminal Capability Ack*. Ils déterminent ensuite qui sera le maître dans la conversation grâce à un échange de messages H.245 *Master / Slave Det / Ack*. Cela sert à résoudre les conflits au cas où les deux terminaux chercheraient simultanément à devenir MC (*Multipoint Controler*) ou à ouvrir l'un vers l'autre et simultanément un canal bidirectionnel.

5.4.2.1.3 Phase C : établissement de la communication audiovisuelle

Les paquets de données voix et image vont circuler dans plusieurs « canaux logiques » H.245. Sauf pour les éventuelles données T.120, ces canaux sont unidirectionnels.

« A » ouvre donc un canal logique vers B pour le son. Il envoie pour cela un message H.245 *Open Logical Channel* qui contient le numéro qui sera attribué au canal H.245 à ouvrir et les paramètres correspondant (numéro de port, type de données (par exemple, audio G.711), et les paramètres additionnels que sont, par exemple pour des données H.225.0, le numéro de session RTP, l'adresse de transport pour les données de retour RTCP (adresse IP + port) unicast ou multicast, le type de données RTP, et si l'émetteur cesse d'émettre pendant les silences).

B renvoie *Open Logical Channel Ack* pour ce numéro de canal H.245, il y mentionne le port UDP où A peut envoyer ses données RTP, et le port TCP où A peut envoyer ses données de contrôle RTCP.

B ouvre à son tour un canal et A confirme.

5.4.2.1.4 Phase D : dialogue

A et B « parlent », les paquets sont échangés sur les canaux virtuels établis précédemment selon le protocole RTP/RTCP repris par H.225.0. Plus précisément, les

données RTP circulent vers le numéro de port précisé dans *Open Logical Channel*, et les données RTCP sur le port suivant.

Il est à noter qu'il y a plusieurs services d'appel applicables qui sont inclus dans les spécifications H.323 comme les variations de la bande passante, le statut et la conférence ad hoc.

La bande passante de l'appel est initialement négociée et fournie par le *gatekeeper* pendant la phase d'initialisation de l'appel, mais à tout moment pendant un appel, le terminal ou le *gatekeeper* peut en demander une augmentation ou une diminution.

5.4.2.1.5 Phase E : fin

Dans l'exemple de la figure 5.7, quand l'un des terminaux souhaite terminer l'appel, il commence par fermer d'abord tous ses canaux logiques pour la vidéo, les données et l'audio, grâce au message H.245 *closeLogicalChannel* correspondant au numéro de canal ouvert et qui sera acquitté par un message *closeLogicalChannelAck*. Par la suite, il envoie un message H.245 *endSessionCommand*, attend de recevoir le même message de son interlocuteur et ferme le canal de contrôle H.245.

Si un canal de signalisation d'appel H.225 a été ouvert, chaque terminal doit envoyer un message *ReleaseComplete* avant de le fermer.

Du moment où un *gatekeeper* est présent, il doit être mis au courant de la libération de la bande passante. Pour cela, les terminaux doivent se désengager de leur *gatekeeper* en utilisant le message *Disengage Request*. Le *gatekeeper* répond alors par un *Disengage Confirm* et l'appel est terminé.

5.4.3 Intégration avec H.323

La section précédente décrit explicitement un appel H.323 effectué entre deux terminaux, soit en présence d'un Gatekeeper ou en son absence. Il existe deux méthodes de sécurité offertes par le protocole de sécurité décrites au chapitre 5 :

- une solution directe entre deux entités communicantes ;
- une autre solution entre 3 ou plusieurs entités communicantes où les terminaux d'extrémité sont les usagers et les entités intermédiaires peuvent être soit des Gatekeeper ou bien des passerelles (Gateways).

Afin d'intégrer SVSP, il est judicieux de choisir la méthode de sécurité adéquate à implémenter. Ainsi, si un appel H.323 ne requiert pas la présence de Gatekeeper, il est recommandé d'utiliser la méthode à deux entités présentée par SVSP.
Si un (ou plusieurs) Gatekeeper existe, et afin de minimiser la charge cryptographique au niveau des terminaux, il est recommandé d'utiliser la seconde méthode de sécurité (à 4 entités) proposée par SVSP.

Une fois la méthode de sécurité a été choisie, nous passons alors à l'intégration de SVSP avec H.323. Trois cas de figures se présentent pour intégrer SVSP dans un appel prenant place dans le réseau IP utilisant le standard H.323:

1. Soit l'appel s'établit entre les deux usagers en mode natif puis à la décision des deux usagers, l'appel bascule en mode sécurisé par SVSP. On se base sur l'appel établi puis on déroule SVSP. ce mode nécessite de signaler l'utilisation entre les entités de l'utilisation de SVSP, puis d'ouvrir une connexion UDP qui

transportera alors les messages de SVSP pour assurer l'authentification des entités et la non-répudiation de l'appel. les usagers auront le choix entre l'utilisation de SVSP pour assurer la sécurisation des médias, ou bien ils pourront négocier l'utilisation d'un autre protocole de sécurité (e.g., SRTP) si les entités communicantes le supportent.

2. Soit l'appel est établi en mode sécurisé SVSP; dans ce cas, SVSP fait partie intégrante du réseau sous-jacent et donc du protocole de signalisation qui accompagne l'application de voix sur IP.
3. la troisième méthode consiste à utiliser le canal *data* pour déployer le protocole SVSP pour assurer la sécurisation de la communication téléphonique entre les deux communicants

Le premier cas de figure consiste à signaler l'utilisation de SVSP dans un message de contrôle H.245. Ceci nécessite d'intégrer la négociation de cette capabilité dans un message *H.245TerminaCapabilitySet*.

La recommandation UIT H.245 est principalement une bibliothèque de messages codés en ASN.1 utilisés dans le canal de contrôle H.245 ouvert au début de l'appel pour négocier des codecs communs, et qui assure ensuite toutes les fonctions de gestion des flux médias. H.245 décrit également les machines à états utilisées dans H.323 et de nombreuses autres recommandations de l'UIT pour la gestion des flux média, particulièrement de la vidéo. H.245 est également utilisée par le standard de visioconférence RNSI H.320, le standard de visioconférence pour lignes analogiques H.324, et le nouveau standard de visioconférence pour téléphones 3G H.324M.

L'utilisation du format ASN.1 (pour *Abstract Syntax Notation 1*) est l'une des raisons qui ont donné à H.323 sa réputation d'être un protocole complexe. Le format ASN.1 est très utilisé dans les applications télécoms, car l'utilisation de compilateurs produisant des codeurs/décodeurs automatiquement améliore de manière spectaculaire la robustesse des applications. L'utilisation de tels outils permet de gagner énormément de temps, et enfin une spécification formelle en ASN.1 permet d'éviter toute ambiguïté dans les spécifications, et facilite donc l'interopérabilité entre applications.

La difficulté sera dans l'ajout de nouveaux objets codés en ASN.1 dans la recommandation H.245 et de l'implémenter dans les équipements déjà existants. Il est possible d'utiliser les paramètres *NonStandard* conçu spécifiquement dans le standard H.323 pour permettre aux vendeurs d'intégrer des protocoles propriétaires avec H.323. C'est le cas d'une proposition qui a été faite pour intégrer le protocole de sécurité FNBDT avec H.323. Mais nous aurons toujours les limitations d'ajouter des lignes dans le code ASN.1 de la recommandation H.245.

Le second cas de figure consiste à intégrer SVSP dans le protocole de signalisation qui est le protocole H.225. Dans la version 4 de H.323, un apport important de cette version est la description de la manière de transporter les protocoles qui n'ont pas de correspondance exacte en H.323 dans les messages standards de H.323 [tunnel H.323]. Les procédures de cette proposition s'appliquent à n'importe quel type d'entité terminale, spécifiquement au Gatekeeper et aux passerelles. Ainsi les tunnels crées sont terminés dans une entité logique appelé « *terminating tunnel* ». Typiquement, ces terminaisons de tunnel sont localisées dans les passerelles qui connectent des parties d'un réseau non-

H.323 avec un réseau H. 323. Si un Gatekeeper est présent dans le réseau H. 323, il peut participer dans le « *tunnelling* » de la signalisation non-H.323. Dans certains cas, la terminaison du tunnel peut être localisée dans un Gatekeeper.
Le contrôle d'appel et les procédures du protocole « tunnellé » sont distincts des états de contrôle d'appel et des procédures du protocole H.225.0 : un point terminal supportant la signalisation par tunnel doit être capable de séparer les deux types de signalisation. Des identifiant et des objets spécifiques sont utilisés dans le codage du protocole H.225 (e.g., le protocole signalé peut être tunnellé et être identifié par le TunnelledProtocol. Les messages de signalisation non-H.323 seront alors « tunellés » dans des messages H.225, spécifiquement les messages « *Setup* », « *Call Proceeding* », « *Alerting* » et « *Facility* », dans les phases A et E de la fig. 5.7.
L'annexe M1 [ITUT-H235-03] décrit l'encapsulation du protocole QSIG de services supplémentaires utilisés principalement par les PBX RNIS, et l'annexe M2 [ITUT-H235-03] décrit l'encapsulation du protocole ISUP (le protocole standard d'établissement d'appels dans les coeurs de réseaux téléphoniques classiques). Nous pouvons nous en inspirer pour intégrer SVSP avec H.323 dans le protocole de signalisation H.225. Cette méthode d'intégration sera utile dans le cas d'une sécurisation suivant la méthode des entités tierces (TAA) présentée dans le chapitre 5.

A chaque version du standard H.323, chaque ajout de fonctionnalités, dont les échanges d'informations étaient décrits explicitement en ASN.1, nécessitait une nouvelle version du fichier de description ASN.1 des messages de contrôle. Un des aspects positifs est que cela garantissait l'interopérabilité des nouvelles implémentations, au moins au niveau de l'analyse des messages, mais la taille croissante de ce fichier ASN.1 posait aussi quelques problèmes aux développeurs ne souhaitant supporter qu'une fraction des fonctions offertes par le standard. Depuis la version 4, un mécanisme d'extension générique permet notamment d'indiquer les fonctions qui sont supportées, requises ou simplement souhaitées, sans nécessiter d'ajouter la syntaxe ASN.1. Ces nouvelles extensions de la version 5 utilisent le cadre d'extension générique (*Generic Extensibility Framework*) [ITUT-H235-03] qui a été introduit en H.323 v4, ce qui a permis de stabiliser les fichiers de description de structures ASN.1 et de mieux intégrer les nouvelles fonctionnalités. Le cadre générique d'extensibilité permet à de nouvelles caractéristiques d'être facilement ajoutées au protocole sans affecter la spécification fondamentale de H.225.0. Le cadre extensible consiste en deux parties :

- Le transport opaque de données dans H.225.0 messages.
- La négociation de fonctions supportées.

Pour éviter de compliquer l'intégration et assurer une transparence totale vis-à-vis du réseau IP et du standard H.323, nous opterons pour la troisième méthode pour intégrer SVSP avec H.323. Cette intégration n'impliquerait aucune modification du standard H.323 ni ne modifierait le codage ASN.1 du protocole de signalisation H.225. Il suffit d'intégrer SVSP sur le canal *data* ouvert de bout en bout entre les deux correspondants une fois que la signalisation d'établissement d'appel H.323 est terminée.

5.4.4 Cas d'une conférence multi parties

Pour effectuer une conférence H.323 à plusieurs parties, une entité spécifique est utilisée qui est le MCU (pour *Multipoint Control Unit*). Cette entité est un point terminal

qui assure le support de conférences multipoints. Le MCU consiste en un MC et l'absence ou la présence d'un ou plusieurs MP.

Un MCU typique qui supporte les conférences multipoints centralisées consiste en un contrôleur multipoint et des processeurs multipoints audio, vidéo et de données. Un MCU peut supporter les conférences centralisées et décentralisées [ITUT-H235-03].

Pour les conférences centralisées, une session est établie en *unicast* entre chaque usager et le MCU. Afin de sécuriser cet appel conférence, une session SVSP par usager est ouverte et gérée par le TAA déployé au niveau du MCU.

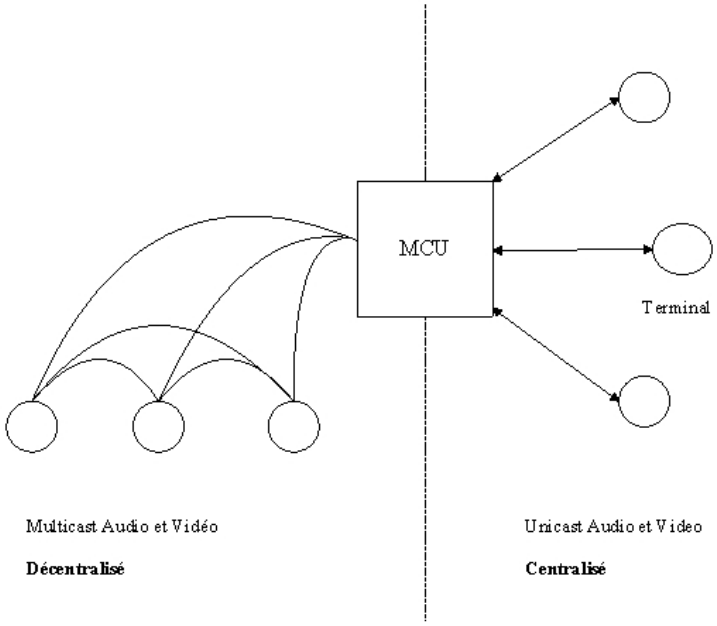

Figure 5.8 - conférence entre usagers H.323 contrôlée par un MCU

Pour les conférences décentralisées, les médias audio et vidéo sont échangés en *multicast* entre les usagers impliqués dans la conférence. Des études futures devraient être effectués pour appliquer SVSP à un contexte *multicast*.

5.4.5 Conclusion

Dans cette section, nous avons détaillé un appel voix sur IP typique avec le standard H.323. Ainsi, nous avons pu remarquer les différentes phases et les entités ainsi que les protocoles impliqués dans l'établissement de l'appel et l'acheminement des informations de médias. Afin de pouvoir intégrer SVSP avec le standard H.323, nous avons proposé deux méthodes pour assurer cette intégration tout en gardant une interopérabilité avec les différents vendeurs. En fait, la première méthode utilise le transport de la signalisation non H.323 dans des messages de signalisation H.225 qui se termine au niveau des passerelles. Cette méthode est utilisable quand un appel voix sur IP est établi avec un autre réseau de téléphonie non-IP. La seconde méthode consiste à utiliser un cadre générique extensible assuré à partir de la version 4 du standard H.323,

qui permet d'introduire de nouvelles fonctionnalités sans toucher au code ASN.1 du protocole de signalisation H.225.
Afin d'assurer une bonne intégration de SVSP dans H.323 sans toucher à ce standard ni aux protocoles qui l'accompagnent, le moyen le plus sûr est de transporter les messages SVSP dans le canal *data* et donc sur le canal RTP ouvert entre les deux interlocuteurs transportant leurs paroles.

5.5 Conclusion générale

Dans ce chapitre, nous avons étudié les possibilités d'intégrer le protocole de sécurité SVSP dans certaines infrastructures de téléphonie les plus répandues et les plus utilisées. Nous avons montré les limitations et difficultés d'intégration au niveau de chaque infrastructure et les solutions qui peuvent les résoudre. Afin de rendre ces propositions valables, des travaux futurs d'implémentations devraient être effectués. Un exemple de cette intégration avec le protocole SIP sera proposé dans le chapitre suivant.

Chapitre 6

6 Analyse et implémentation de SVSP

6.1 Résumé

Dans ce chapitre, nous analysons les services offerts par le protocole SVSP en étudiant leurs avantages et inconvénients. Une implémentation d'un prototype de SVSP est présentée, codée avec le langage de programmation Java. Une autre intégration de SVSP avec le protocole de voix sur IP SIP est implémentée en utilisant la plate-forme JAIN-SIP.

6.2 Introduction

Le protocole de sécurité SVSP a été défini pour satisfaire les exigences de l'architecture de sécurité spécifiée pour la téléphonie. Cependant, comme tout protocole de sécurité, il a ses avantages et ses limitations. Une analyse des différentes entités procédurales et fonctionnelles du protocole permettra de mieux cibler ses vulnérabilités. Cette analyse sera suivie de l'implémentation d'un prototype de SVSP présentant certains services de sécurité offerts par l'architecture de sécurité, essentiellement une authentification entre deux communicants basée sur les certificats auto-signés, la confidentialité de la communication par chiffrement symétrique, l'intégrité des messages échangés par une fonction de hachage unidirectionnelle et finalement la génération d'un rapport de non-répudiation signé par chaque utilisateur. Une intégration de ce prototype dans un appel voix sur IP basé sur le standard SIP est présentée en utilisant la plate-forme JAIN-SIP de Java.

6.3 Analyse de SVSP

Le protocole SVSP est composé de plusieurs entités fonctionnelles et procédurales afin de satisfaire le déploiement de la sécurité de la téléphonie et de respecter les exigences de cette dernière tout en assurant une abstraction des couches basses de la pile protocolaire. Dans ce qui suit, nous analysons chaque entité à part en se basant sur la définition du protocole SVSP dans le chapitre 4.

6.3.1 Les cartes intelligentes

Une carte intelligente (pour *Smart Card*) ressemble à une carte de crédit seulement avec un fragment enfoncé dans son plastique. Ce fragment est en fait un petit ordinateur complet avec une mémoire non volatile, une partie pour l'emmagasinage des données, un système d'opération de la carte (le COS pour *Card Operating System*) et les protocoles de communication qui l'accompagnent.

Quels sont les risques d'utilisation des cartes intelligentes?

- Perte de la carte : Si une carte intelligente est perdue ou est volée, les informations emmagasinées sont perdues. Afin de sécuriser l'accès aux cartes à puce, nous pouvons soit le sécuriser par un mot de passe (PIN code) ou bien par une empreinte digitale (Biométrique). Dans le second choix, pour que l'accès soit valide lors de l'insertion de la carte, une comparaison entre l'empreinte digitale (*fingerprint*) mémorisée dans la carte et celle de l'utilisateur est effectuée. Cette empreinte digitale peut servir aussi comme identifiant de l'utilisateur dans le déroulement du protocole SVSP lors de l'émission de la carte par l'entité de certification (TAA).
- Information confidentielle compromise : les cartes qui seront utilisées pour assurer la sécurisation de la téléphonie portent certaines informations confidentielles (identifiant de l'usager, clé pré-partagée, certificat du TAA, etc.). Si ces informations ont été compromises, elles peuvent être utilisées pour escroquer les usagers et compromettre la sécurité des appels. Un soin extrême doit être pris pour garder ces informations confidentielles et bien protégées.
- Attaques physiques : Les attaques physiques tentent de renverser l'ingénierie de la carte et de déterminer les informations tenues secrètes (les clés). Ces attaques ont été démontrées en pratique contre les cartes commerciales, notamment par Ross Anderson de Cambridge et Marcus Kuhn de Purdue [JavaCardSecurity].
- Génération des nombres aléatoires : les cartes intelligentes utilisées pour des besoins de sécurité exigent des fonctions cryptographiques câblées sur la carte. La cryptographie (i.e., RSA ou AES) nécessite la génération de nombres aléatoires et exige des cartes la capacité de les produire. Produire des nombres pseudo-aléatoires n'est pas une tâche facile, surtout si les ressources sont limitées sur la carte. Un soin particulier doit être pris pour que la production de ces nombres soient de haute qualité, afin d'éviter de compromettre les systèmes de crypto.
- Attaques aux protocoles: des analyses cryptographiques par méthode formelle sur les protocoles que peuvent contenir les cartes intelligentes ont été effectuées afin de vérifier et de valider [McG98] la sécurité des cartes et de spécifier la conception de rigoureuses analyses. En plus de l'analyse de protocole, Bruce Schneier, l'auteur de Cryptographie Appliquée [Sch96], a identifié récemment une famille d'assauts basés sur l'utilisation multiple de protocoles sur une carte pour déterminer l'information secrète. Beaucoup de cartes intelligentes offrent à de multiples protocoles l'utilisation d'une même clé secrète. Il y a un risque que l'assaut de protocoles multiples ne compromettra la sécurité de carte.
- Les problèmes dus au terminal : finalement, les cartes intelligentes doivent être capables d'afficher des informations de leurs contenus et de certaines transactions à l'usager. Cet affichage doit être inattaquable et digne de confiance. L'usager doit se fier aux informations données par la carte. Ainsi, l'utilisation d'une carte intelligente sur un PC non sécurisé peut former un risque élevé aux informations contenues dans la carte.

Les nouvelles cartes déployées sont capables de fonctions de PKI [setec1], ceci évite de devoir exporter la clé privée à l'application résidant sur le terminal. Par exemple lors de l'envoi d'un courrier signé, les applications de courrier produisent premièrement un hachis du document écrit et commencent la communication avec la carte. L'application envoie la valeur du hachis (condensât) à la carte qui le signe avec la clé privée dans la carte. De cette façon, la clé privée n'est jamais exportée au public (l'ordinateur).

6.3.2 Choix du vocodeur

Certains réseaux de téléphonie ne permettent pas d'accéder au codec de voix utilisé dans les équipements téléphoniques comme nous l'avons vu dans la partie d'études d'intégration de SVSP dans le chapitre 5. Dès lors, le protocole de sécurité SVSP offre son propre vocodeur. Mais le choix du *codec* de la parole doit s'effectuer en fonction de la qualité et du débit de transmission sur la ligne téléphonique pour éviter d'occuper toute la bande passante disponible. Ainsi plusieurs facteurs affectent le choix du vocodeur qui dépend essentiellement des réseaux de téléphonie traversés et des débits en ligne disponibles. Ce choix doit prendre en considération plusieurs facteurs et assurer un compromis entre une bonne qualité de la voix et une sécurisation de la téléphonie.

Le fait d'utiliser des codecs de voix à débits faibles au niveau applicatif permet de réduire la latence et la gigue. Aussi on peut envisager l'utilisation de DSP (*Digital Signal Processors*) qui améliorent la qualité de la voix dans une implémentation de sécurité par le protocole SVSP.

6.3.3 Les limitations de SVSP

Nous présentons ci-après certaines limitations qui peuvent être attribuées au protocole de sécurité SVSP et qui sont comme suit :

1. Les contraintes d'une bonne transmission de la communication téléphonique imposent des exigences pour une transmission temps réel assurant un délai de propagation de bout en bout ne dépassant pas les 150 ms, des taux de pertes inférieur à 1% et une gigue (ou variation de délai) inférieure à 1 ms. L'inconvénient majeur de cette proposition réside essentiellement dans la lourdeur des opérations cryptographiques par clés publiques, essentiellement si elles étaient déployées sur des postes téléphoniques ayant de faible puissance de calcul. Ce qui introduiraient des délais supplémentaires et nécessiteraient le changement des équipements (téléphones) existants.

2. Une limitation avec le modèle d'architecture de sécurité à deux entités est due à la gestion des certificats et des listes de certificats révoqués (CRL) au niveau des usagers. Chaque usager doit s'assurer de la validité des certificats des correspondants avec qui ils communiquent afin de les authentifier. Il est possible de permettre à l'usager de sauvegarder dans son espace d'emmagasinage spécifié par le module *Certificat* du protocole SVSP le certificat de son correspondant validé la première fois avec le GTAA pour une période de temps déterminée. Une fois que cette période est écoulée, l'opération de validation doit être effectuée de

nouveau par l'entité responsable. L'utilisation d'une GTAA pour valider les certificats augmente les délais d'établissement d'une session de sécurité par SVSP mais permet de se parer contre l'attaque du MitM.
Afin de minimiser la charge de l'utilisateur, il est recommandé de reporter l'utilisation des certificats aux TAAs et la liste CRL au GTAA et donc d'appliquer le modèle d'authentification avec les clés pré-partagées entre l'utilisateur et son TAA de rattachement.

3. Le protocole SVSP introduit un *overhead* de 28 bytes si l'extension d'entête n'est pas utilisée pour le transport des informations sur les paramètres de sécurité et certains messages spécifiés dans la définition du protocole SVSP. Le tableau suivant compare SVSP avec SRTP et IPsec en terme *d'overhead* protocolaire. Il est évident que l'utilisation de SRTP dans un appel voip de bout en bout est un peu moins coûteuse en terme *d'overhead* que SVSP.

Table 5 – comparaison de *l'overhead* introduit par SVSP et autres protocoles de sécurité

Type de Paquet	**Entête (octet)**	**Taille du payload (octet)**	**Taille total du paquet (octet)**	**Ratio (payload/taille totale)**
IP/UDP/RTP	40	40	80	50 %
IP/UDP/RTP + SRTP (4 bytes)	44	40	84	47.7 %
IP/UDP/RTP + SRTP (10 bytes)	50	40	90	44.4 %
IP/UDP/RTP + SVSP (28 bytes)	68	40	108	37 %
IP/UDP/RTP + IPsec (ESP avec DES)	90	40	130	30.7 %
IP/UDP/RTP + IPsec (ESP avec 3DES+SHA)	110	40	150	26.6 %

Quand un message SVSP transporte des informations de voix, pour éviter d'avoir un *overhead* énorme, il est recommandé d'éviter l'utilisation des extensions d'entête. Ces dernières seront utilisées uniquement durant les phases d'authentification et de non-répudiation qui ne transportent pas des données de voix.

4. Limitation de la mémoire des cartes intelligentes :

La carte intelligente possède plusieurs composants qui sont :

- Une mémoire non volatile (ROM) qui contient l'identité de l'usager et sa clé pré-partagée ainsi que d'autres informations spécifiques à l'usager comme le numéro de série de la carte et le nom de l'utilisateur. Sa capacité varie de 2 KB à 16 KB. Cette capacité ne peut pas contenir une clé de 256 bits ;

- Une mémoire électriquement effaçable (EEPROM) qui contient les données qui changent avec le temps comme le certificat de l'usager, l'API et l'application SVSP si sa capacité est suffisante. Cette mémoire peut être protégée par un code PIN. Sa capacité varie de 2KB to 32 KB ;
- Une mémoire dynamique (RAM) qui contient les informations relatives à chaque session SVSP comme les clés dérivées, l'identifiant de la session, etc. Sa capacité varie de 128 Bytes à 512 Bytes ;
- Un processeur (CPU) ;
- Des interfaces d'entrées/sorties ;
- Une logique de sécurité pour sécuriser les échanges entre les différents composants de la carte.

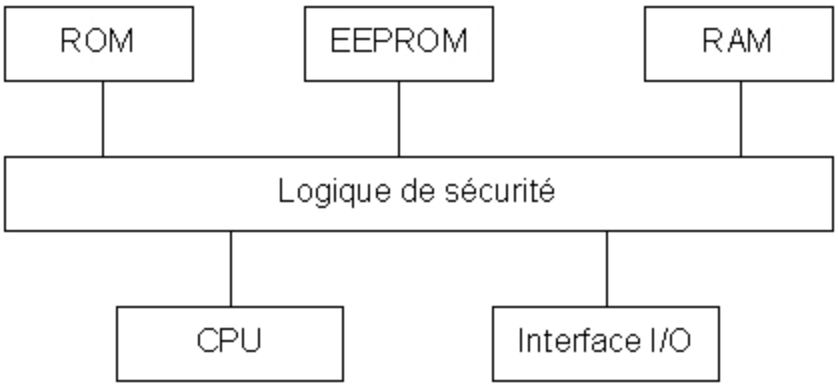

Figure 6.1 - Architecture d'une carte à puce

Une carte Java est une carte intelligente qui peut interpréter le code d'octet de Java, similaire aux navigateurs de Java. Cette carte est basée sur un sous-ensemble d'une API Java. Pour être capable d'exécuter ce code, la carte doit avoir au moins une capacité de 16K de mémoire morte, 8K de EEPROM et 256 octets de mémoire dynamique. Il existe actuellement sur le marché de telles cartes [Sun] et sont en continuelle évolution. Le nombre de nouvelles applications possibles tournant sur une carte intelligente Java est en évolution. Cependant, les contraintes de mémoire et des interfaces des cartes intelligentes affecteront profondément le style de programmation.

Quels sont les risques apportés par les Smart Card Java?
Un des plus grands problèmes dans la sécurité de Java est de conserver la sûreté en même temps que le chargement dynamique des classes. Si on arrive à confondre la machine virtuelle avec les types d'objets qu'elle manipule, on pourra alors casser le modèle de sécurité de la carte. Des méthodes pour sécuriser les cartes Java sont discutées dans [JavaCardSecurity]. Les cartes Java permettent à de multiples applications de résider sur la même carte intelligente et d'assurer leur sécurité [JavaCardSecurity]. Bien que les cartes Java définissent « des cloisons pare-feu d'applet » pour protéger les applets l'un de l'autre, il y a un risque d'assauts inter-application. Les cloisons pare-feu doivent être parfaitement appliquées pour permettre l'usage sûr d'applets multiples.

6.4 Mise en oeuvre de SVSP

La mise en œuvre de SVSP porte sur deux parties : un prototype de SVSP et une intégration de ce prototype avec le protocole SIP pour effectuer une communication voix sur IP sécurisée. Le choix de la plate-forme de programmation utilisée est tombée sur Java puisqu'il est utilisé pour développer des applications d'entreprises de grandes envergures et des applications pour les équipements de consommation (e.g., les téléphones IP, les mobiles, les PDAs,…). La plate-forme JAIN-SIP permet de séparer l'application de la pile protocolaire SIP en normalisant l'interface et la sémantique. Elle permet aux développeurs d'accéder au protocole SIP implémenté conformément au RFC 3261.

6.4.1 Implémentation du prototype SVSP

Dans ce paragraphe, nous présentons en détail l'implémentation d'une session SVSP entre deux utilisateurs Alice et Bob. Nous considérons que Alice va ouvrir une connexion de transport TCP/UDP avec Bob dont elle connaît l'adresse IP et le numéro de port.
Les messages SVSP seront encapsulés dans des segments TCP /UDP constituant la charge utile (*payload*) de ses segments.

Chaque message SVSP est formé d'une entête suivie d'une entête d'extension et d'un champ utile, le tout protégé par un champ MAC ou bien une signature digitale afin d'assurer l'intégrité du message SVSP. Quelques champs de l'entête SVSP auront une valeur fixe pour tous les messages échangés, nous notons les champs suivants:
Le champ *Version* sur 2 bits indique la version du protocole SVSP, et aura la valeur (1). Le champ *SACK* qui porte sur 1 bit n'est pas utilisé dans cette implémentation du protocole, il sera mis à (0). Le champ *Voice ID* s'étend sur 4 bits et indique le type de protocole de voix utilisé durant la communication. Pour ce champ, nous allons définir des valeurs fixes suivant le type de réseau de télécommunication utilisé par l'émetteur du message SVSP, et ceci comme suit:

Table 6 – valeurs du champ VoiceID du messages SVSP

RTC	0
ISDN	1
GSM	2
UMTS	3
SIP	4
H323	5
WIFI	6
Autre	7

Le champ *Session ID* formé de 8 bits contient un numéro qui peut être généré d'une manière aléatoire par chaque entité et qui sera utilisé le long de la session par chaque entité pour identifier la session.
Les autres champs du message SVSP changent de valeurs pendant l'échange des messages, nous notons :

Le champ *EHP* (1 bit) change de valeur selon que le message SVSP contient une entête d'extension ou pas. Il prendra la valeur 1 pour indiquer la présence de champs d'extension d'entête, sinon il sera mis à 0.
Le champ *Total Length* indique la longueur du message, sans le champ MAC.
Le champ *Sequence number* porte le numéro de séquence du message. La valeur initiale de ce champ est d'abord générée d'une manière aléatoire par l'application et puis incrémentée de un pour chaque nouveau message envoyé.
Le champ *Checksum* sera calculé sur chaque paquet SVSP. Il est essentiellement utilisé sur des liaisons non fiables.
Le champ *Timestamp* peut contenir tout simplement le temps d'horloge de la machine.

Suite à cette description détaillée de l'entête du message SVSP, considérons maintenant l'échange entre Alice et Bob; Alice va initialiser la session SVSP en envoyant à Bob un message *U-RUCA.req*. Ce message va contenir en plus du champ utile deux entêtes d'extensions ; le premier *extension header* contient la certification de Alice, et le second contient le *User Capability Set* de Alice.

Pour la génération des certificats et pour des fins spécifiques à ce prototype, nous utiliserons des certificats auto-signés (*Self signed certificat*) par les utilisateurs eux-mêmes et cela purement pour simplifier l'implémentation qui est une manière plus efficace que d'obtenir des certificats signés par des autorités de certification.

Le message *User Capability Set* contient la liste des noms des algorithmes de cryptographie supportés par Alice, citons à titre d'exemple :

- DES pour le chiffrement/déchiffrement ;
- HMAC avec SHA1 : pour générer des Mac tags ;
- SHA1 ;
- Diffie_Hellmen pour l'algorithme de génération des clés de sessions.

Le dernier champ du message contient la signature de Alice portée sur tout le message SVSP.
Le champ utile va contenir un nombre aléatoire *Random(Alice)*. Ce nombre doit être échangé d'une manière secrète, si Alice avait effectué des sessions SVSP précédentes avec Bob alors le certificat de Bob est connu par Alice. Ainsi Alice connaît la clé publique de B et le *Random(Alice)* sera chiffré par cette clé. Si c'était la première fois que Alice communique avec Bob, alors Alice ne possède pas le certificat de Bob et par la suite ne connaît pas encore la clé publique de B. Le champ utile contiendra alors une indication que le *Random(Alice)* sera transmis dans un message supplémentaire après l'obtention du certificat de B et sa validation. On peut illustrer cela comme suit :

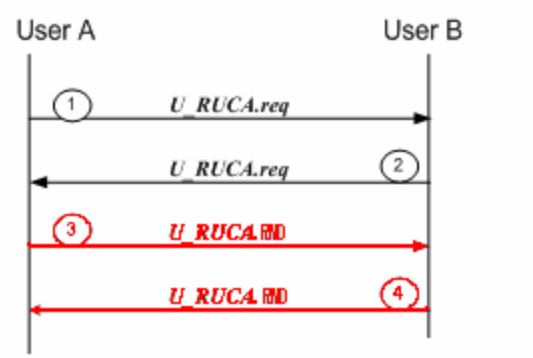

Figure 6.2 – scénario d'échange de message SVSP contenant un *Random*

Suite à cet échange de messages nécessaires pour assurer une authentification mutuelle entre les entités et la négociation des paramètres de sécurité, Alice connaît maintenant le *Random(Bob)*, la clé publique de Bob et le *User Capability Set* de Bob et vice versa qui ont été échangés d'une manière secrète et fiable. Alice et Bob peuvent générer localement une clé secrète commune que nous nommerons la *Master-Key* en utilisant un algorithme approprié comme Diffie_Hellman par exemple. Cette clé sera utilisée pour le chiffrement des messages SVSP suivants et les données utiles des médias ainsi que pour générer des HMACs sur les messages SVSP. Il est recommandé de dériver une clé à partir de cette *masker-key* pour chacune des fonctions citées précédemment.
Alice chiffre son identité ID_A avec la clé publique de Bob. En utilisant une fonction de hachage unidirectionnelle qui prend comme paramètres ID_A et *Random(Bob)*, Alice génère un $HRES=hash(RNDB, ID_A)$. Elle envoie à Bob un message *U_RUCA.rep* dont le champ utile contient la clé générée $K_{f,A}=f(K_{Bpub}, ID_A)$ et les deux champs d'extensions qui contiennent :

- Le *HRES* ;
- Le *USERCAPABILITYSETACK* ;
- Un paramètre nécessaire pour calculer la clé secrète (dépend de l'algorithme utilisé).

Le même message sera envoyé de Bob vers Alice. Ainsi, Alice et Bob peuvent s'identifier l'un à l'autre en déchiffrant le *ID* reçu et en calculant le condensât de cet *ID* avec la fonction de hachage utilisée et comparant le résultat obtenu avec le *HRES* envoyé par l'autre entité.

Alice envoie à Bob le message *U_RUCA.conf* contenant le *Token(Alice)* dans le champ utile du message et un champ d'extension contenant sous forme de texte :

- *Session ID* ;
- *Cverify* ;
- *SetCipherSpec* pour confirmer l'algorithme de chiffrement qui sera utilisé avec le *master-key*.

Le *Token* sera constitué par l'identifiant de Alice (ID_A) et le *Random(Alice)* tous les deux chiffrés par le *master-key*.

Bob envoie à son tour un message identique *U_RUCA.conf* à Alice. Ces deux usagers peuvent commencer un échange de média sécurisé en utilisant le protocole RTP encapsulé dans un segment UDP avec des numéros de port bien définis.
Supposons maintenant que Alice va terminer la session, elle envoie à Bob un message *U_ESMS.req* qui contient tout simplement une entête d'extension indiquant son besoin de terminer la session. Bob envoie alors à Alice ce même message pour acquitter celui de Alice.
Alice envoie le message *U_ESMS.ind* qui contient dans son champ utile le *Token(Bob)*, une entête d'extension de type *CloseChannel* contenant le *SessionID* de Alice. Bob enverra le même message *U_ESMS.ind* à Alice.

Enfin Alice génère un rapport de non-répudiation (*Message Report*) qui contient :

- La date de l'appel ;
- La duré de l'appel ;
- Le SessionID ;
- Un « Serial Number » ;
- ID_A et ID_B ;
- Le type de l'appel (RTC, GSM…).

Ce rapport sera signé par Alice puis envoyé dans le champ utile du message *U_ESMS.conf*. La signature sera envoyée dans une entête d'extension.
Bob envoie à son tour un message *U_ESMS.conf*, le canal RTP sera fermé ainsi que le canal de transport TCP ou UDP. Les rapports seront sauvegardés dans des fichiers, ces rapports seront utiles pour assurer la non répudiation.
Ainsi, cette implémentation simple permet d'effectuer un échange média sécurisé entre deux entités à l'aide d'une signalisation SVSP et indépendamment de l'infrastructure utilisée pour la communication.

Les entêtes d'extension comportent plusieurs champs ; nous attribuons à chaque champ Type sur 4 bits et Code sur 8 bits les valeurs indiquées dans le tableau suivant :

Table 7 – tableau des valeurs des champs de l'entête d'extension

Paquet	Payload de l'entête d'extension (HE)	Type	Code
U_RUCAreq	User Certificat	0	0
	UserCapabilitySet	0	1
U_RUCArep	HRES	1	0
	UserCapabilitySetAck	1	1
	DiffieHellmanPublicKey	1	2
Paquet	HE payload	Type	Code
U_RUCAconf	« SID+Cverify+SetCipherSpecAck »	2	0
U_ESMSreq	_____	3	0
U_ESMSrep	_____	4	0
U_ESMSind	SessionID	5	0
U_ESMSconf	Signature	6	0

Pour le champ Réservé (4 bits), nous utiliserons un bit (*flag*); ce bit sera mis à *1* pour indiquer qu'il y a une autre entête d'extension qui suit ou à *0* pour indiquer que c'est la dernière entête d'extension.

Le prototype SVSP est implémenté en Java. Pour chaque message SVSP, nous avons créé deux classes, une qui a pour rôle de construire le message, et une autre pour lire ce message. Ces classes feront la vérification sur le *Checksum* et le *Mac tag* pour assurer l'intégrité, l'extraction les informations utiles du champ *payload* et les entêtes d'extension du message SVSP.
Ensuite, nous avons deux autres classes ***User Agent Client*** et ***User Agent Server***. L'initiation de la session SVSP sera effectuée par la classe ***User Agent Client*** qui transmettra le message *U_RUCA.req* vers le destinataire puis nous passerons au scénario déjà décrit précédemment.
Les deux classes Java suivantes, ***Send Message*** et ***Receive Message*** assureront le cryptage et le décryptage des messages textes échangés (pour un service de messagerie).

Pour la voix sur IP, nous utiliserons la librairie *Java Media Framework* qui offre des moyens pour capturer, coder et envoyer la voix sur le réseau en utilisant le protocole RTP.
Afin de faire le chiffrement, le déchiffrement et l'encapsulation des paquets de voix dans des paquets SVSP avant de les transmettre, nous avons implémenté deux nouvelles classes dans la librairie : ***Customg723packetizer*** et ***Customg723javaDecoder*** qui utilise le codec G.723 à 5 ou 6 kbps.
Les différences classes mises en jeu peuvent être illustrées comme suit :

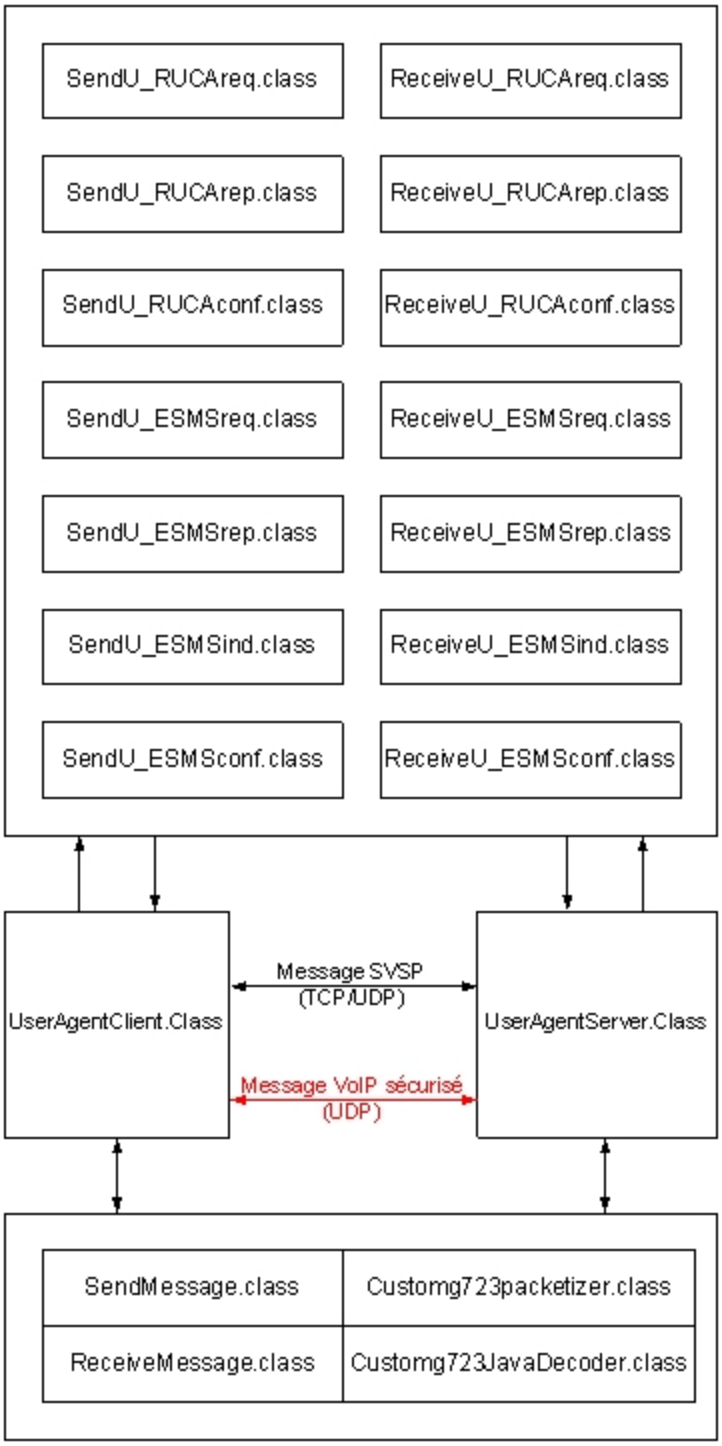

Figure 6.3 – interaction entre les différentes classes Java

Pour la génération des paires de clés et des certificats X.509, on a utilisé l'outil *Keytool*. Cet outil permet de générer une paire de clé (publique, privé) et un certificat associé à la clé publique ; le tout est sauvegardé dans un fichier spécial, nommé par défaut **.***Key Store*. Ce fichier peut contenir plusieurs paires de clés et les certificats

correspondants ; il est protégé par un mot de passe. Il permet aussi d'exporter les certificats dans des fichiers **.***Cer*.
Ainsi, dans notre application Java, nous pouvons accéder au fichier **.***Cer* pour lire et transmettre des certificats, et au fichier **.***KeyStore* afin d'accéder aux clés privées pour faire la signature et le cryptage.

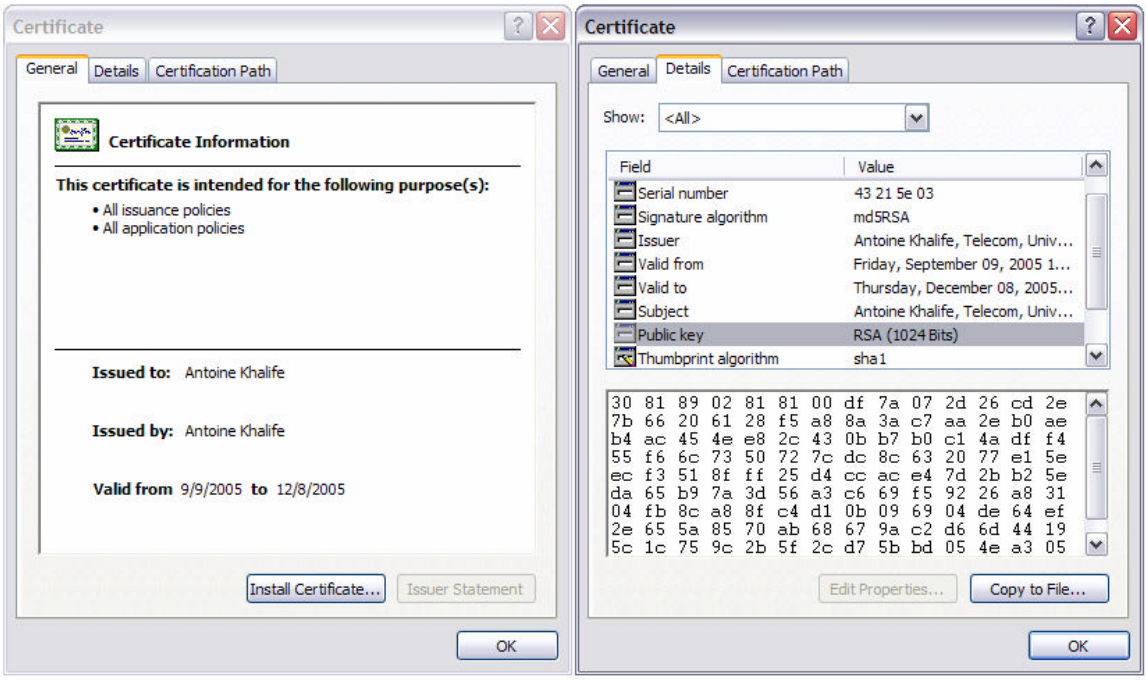

Figure 6.4 - certificat X.509

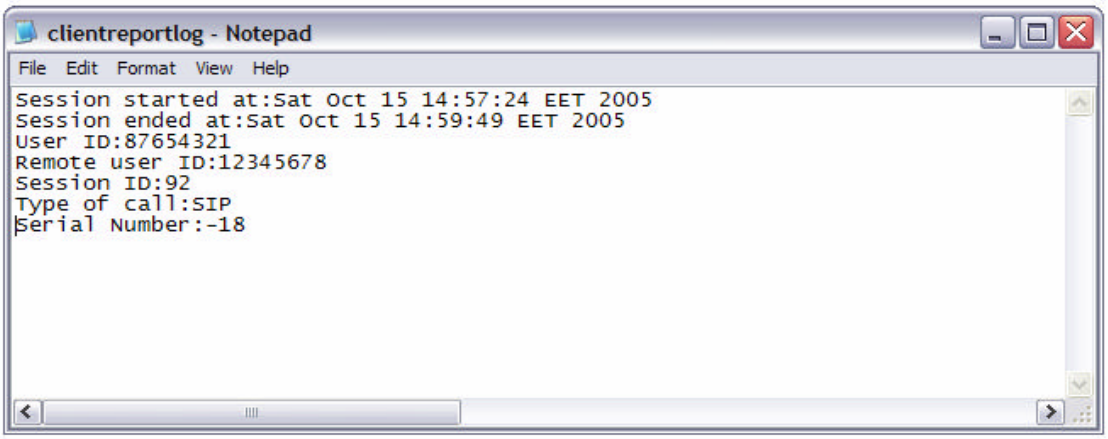

Figure 6.5 – rapport de non répudiation

Table 8 – spécification technique et mesure des paramètres de performance

<table>
<tr><td>Plateforme</td><td colspan="3">JDK5.0, WinXpSP2</td></tr>
<tr><td>Processeur</td><td colspan="3">Intel Centrino 1.5Ghz</td></tr>
<tr><td>RAM</td><td colspan="3">512MO</td></tr>
<tr><td>Disque dur</td><td colspan="3">60GO</td></tr>
<tr><td>Réseau</td><td colspan="3">IP</td></tr>
<tr><td colspan="2">Longueur clé publique/privée</td><td colspan="2">1024bits</td></tr>
<tr><td colspan="2">Longueur clé symétrique</td><td colspan="2">64bits</td></tr>
<tr><td colspan="2">Temps moyen nécessaire pour commencer la session sécurisée (moyen calculé sur 10 mesures)</td><td>1677,5ms (TCP)</td><td>1451,2ms (UDP)</td></tr>
<tr><td colspan="2">Temps moyen nécessaire pour terminer la session sécurisée (moyen calculé sur 10 mesures)</td><td>499,8ms (TCP)</td><td>548,8ms (UDP)</td></tr>
<tr><td rowspan="2">Capacité en mémoire (Service Messagerie)</td><td colspan="2">Application serveur</td><td>13,760KO</td></tr>
<tr><td colspan="2">Application client</td><td>18,288KO</td></tr>
<tr><td rowspan="2">Capacité en mémoire (Service Voix sur IP)</td><td colspan="2">Application serveur</td><td>24,176KO</td></tr>
<tr><td colspan="2">Application client</td><td>26,428KO</td></tr>
<tr><td colspan="2">Délai moyen introduit par le paquetiseur de voix (calculé sur 25 paquets de voix)</td><td colspan="2">0,8ms</td></tr>
<tr><td colspan="2">Délai moyen introduit par le décodeur de voix (calculé sur 25 paquets de voix)</td><td colspan="2">3.2ms</td></tr>
<tr><td colspan="2">Débit du codec G.723 (avant l’application de SVSP)</td><td colspan="2">5 à 6 kbps</td></tr>
<tr><td colspan="2">Débit du codec G.723 sécurisé par SVSP</td><td colspan="2">10 à 11 kbps</td></tr>
</table>

Ainsi le protocole SVSP assure un service de sécurité pour la téléphonie quelque soit le type des infrastructures utilisées, qu'elles offrent elle-mêmes de la sécurité ou pas et quelque soit le codage de voix qu'elles supportent.

Remarques :

Pour optimiser l'implémentation du prototype SVSP, nous proposons le suivant :

- Transmettre les messages SVSP dans des paquets UDP, ce qui minimise le nombre de paquets échangés entre les deux entités de communication pour éviter la phase de *HandShake* de TCP puisque SVSP assure un mécanisme de retransmission en cas d'erreur.

- Utilisation des TAAs pour les réseaux où les terminaux d'extrémité sont à faible capacité de traitement et à bande passante limitée (les TAAs seront des serveurs implémentés au cœur du réseau correspondant) comme c'est le cas avec les réseaux GSM et RTC.

- Utilisation des certificats X.509 pour les réseaux où les terminaux d'extrémité possèdent de la capacité de traitement et de stockage, et où la bande passante est suffisante, ce qui est le cas pour le réseau IP.

6.4.2 Automate du protocole SVSP

Le protocole SVSP offre deux méthodes d'authentification. L'implémentation du prototype SVSP utilise le modèle d'architecture à deux entités et repose sur les certificats. La deuxième méthode utilise une clé pré-partagée entre l'utilisateur et une entité de confiance. Cette méthode est modélisée par deux automates décrivant les procédures d'implémentation de SVSP vues de l'émetteur et du récepteur.

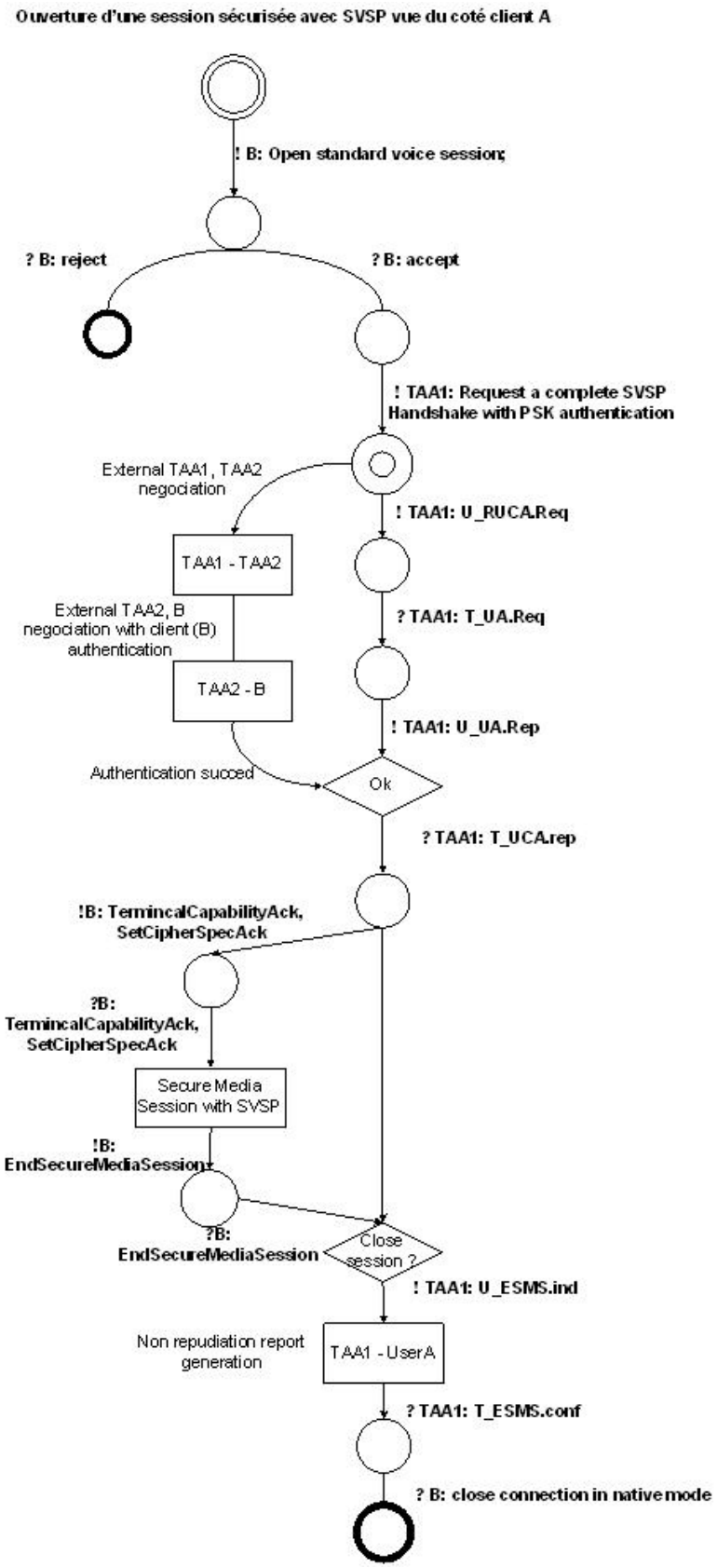

Figure 6.6- Automate SVSP vu du client initiateur de la session

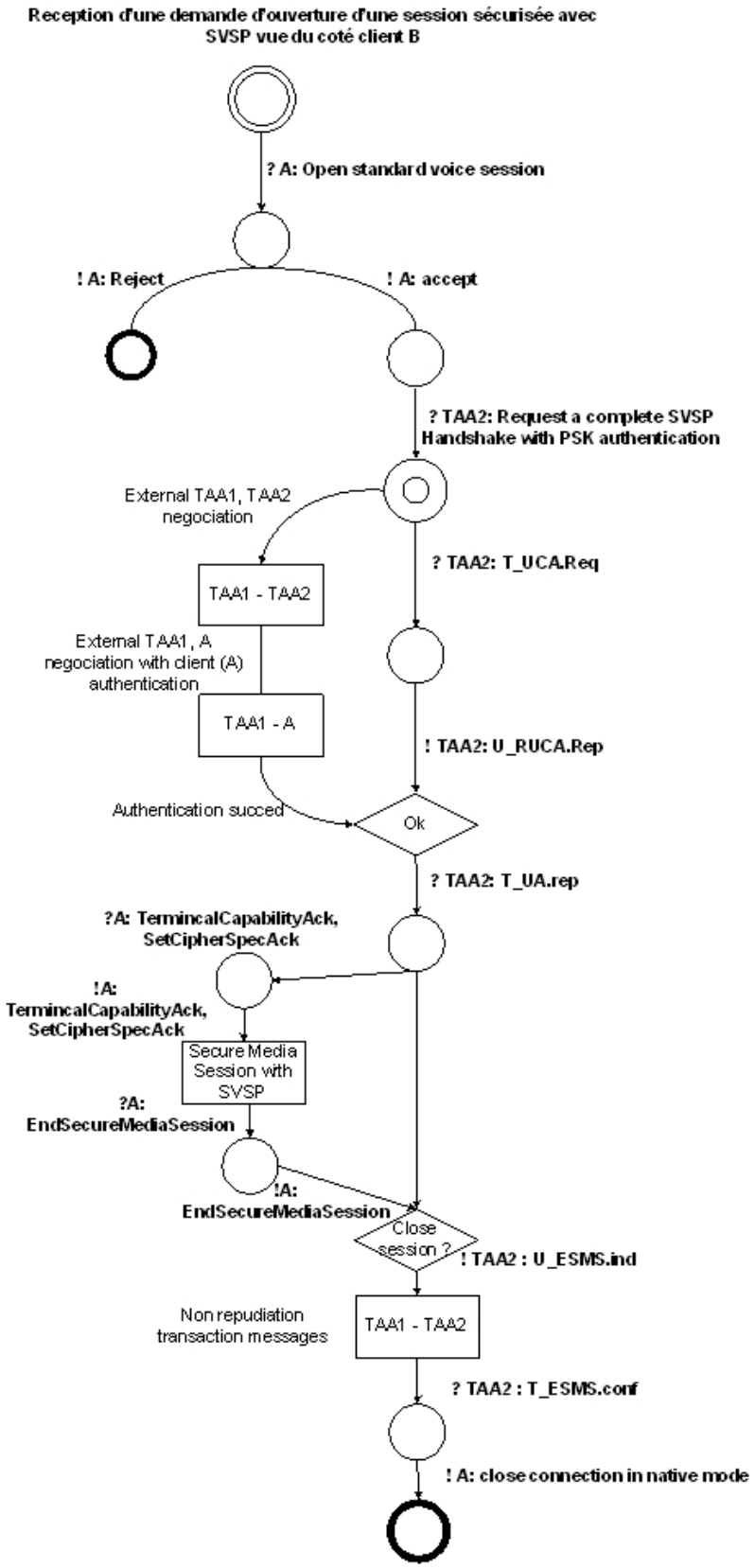

Figure 6.7- Automate SVSP vu du client récepteur

6.5 Intégration de SVSP avec SIP

Comme nous l'avons vu dans le chapitre 2, SIP [RFC 3261] est un protocole de signalisation qui permet de contrôler l'établissement et la terminaison d'une communication voix sur IP. Les messages SIP sont des messages textes plus faciles à traiter que ceux utilisés dans d'autres protocoles de VoIP.

6.5.1 Les vulnérabilités de SIP

SIP présente les mêmes vulnérabilités que les autres protocoles de VoIP mais SIP possède plusieurs facteurs qui le rend moins sécurisé et qui sont :

- La maturité – SIP est relativement un protocole récent et son implémentation est récente aussi ;
- La complexité – SIP est modérément complexe, mais toutes les extensions qui l'accompagnent le rendent comme un protocole complexe ;
- L'extensibilité – SIP supportent des extensions au RFC principal qui sont nouvelles et ont des lacunes de point de vue sécurité ;
- Codage – SIP utilise des messages textes qui sont faciles à capturer et à analyser par un sniffer.

SIP offre une sécurité limitée et assume une interopérabilité entre les vendeurs de produits voix sur IP. L'interopérabilité cause un problème à la sécurité puisque tous les produits ne supportent pas les mêmes standards de sécurité.
Plusieurs implémentations de SIP utilisent le protocole UDP qui n'assurent pas la retransmission des segments perdus ni possèdent des numéros de séquence, ce qui permet à un attaquant de capturer les paquets UDP.
Les attaques qui sont conduites sur le protocole SIP se résument comme suit :

Terminaison d'une Session SIP

La terminaison d'une session SIP se présente quand un attaquant observe la signalisation d'un appel et envoie des messages *bye* aux deux agents utilisateurs de SIP qui les forcent à terminer la session SIP. Les agents SIP ne requièrent pas une authentification forte pour éviter ce genre d'attaques. Un déni de service (DoS) est le premier effet d'une terminaison d'un appel SIP d'une façon inappropriée. Les messages SIP *re-invite* peuvent être utilisés pour modifier une session de média. La redirection des médias vers des adresses *broadcast* peut être la cause d'une attaque DoS.

Détournement d'enregistrement

Un détournement d'enregistrement arrive quand un agresseur imite un agent UA valide, ce connecte à un Registrar SIP et remplace l'enregistrement du UA avec sa propre adresse. Cet assaut cause la redirection de tous les appels venants de l'extérieur vers l'agresseur. Le détournement d'enregistrement peut avoir pour résultat la perte des appels du UA légitime, qui peut être une ressource critique (e.g., le *media gateway*, le Standardiste Automatisé (AA), le système de messagerie vocale). Aussi, le UA escroc peut recueillir

l'authentification ou autre clé. L'escroc peut aussi causer une attaque du MitM capable de recueillir et modifier les messages de signalisation SIP.

Imitation du Proxy

L'imitation du proxy arrive quand un agresseur trompe un agent UA ou un proxy et les amène à communiquer avec un proxy escroc. Ainsi, il aura accès à tous les messages SIP, ce qui lui donne un contrôle total de l'appel. Il s'insère alors dans la signalisation et cause les attaques de *Domain Name Service (DNS) spoofing* et, *Address Resolution Protocol (ARP) cache spoofing*.

Changement des messages

Cette attaque arrive quand un escroc arrive à intercepter et modifier les paquets échangés entre les entités du protocole SIP. Elle peut aussi arriver suite à un détournement d'enregistrement, imitation du *proxy* ou un assaut sur n'importe quel composant fiable de traiter les messages SIP, tels que le *proxy*, le *media gateway* ou la cloison pare-feu (*firewall*).

SIP Spam

Le *spam* est un fléau qui a influé sérieusement les systèmes de e-mail (SMTP). Il a aussi le potentiel de causer des dommages sérieux à la communauté de SIP. C'est un problème qui doit être adressé proactivement et d'urgence.
Principalement, trois manifestations distinctes de *spam* de SIP ont été identifiées et classifiées : le *Spam* de Téléprospecteur, qui forme une série non sollicitée de demandes tentant d'établir une séance de communications; le *Spam* de IM (pour *Instant Messaging*), qui est une série de demandes similaires au *spam* du *e-mail*; et le *Spam* de Présence, où les messages sont envoyés en une tentative pour obtenir une entrée sur la liste d'un utilisateur pour lui envoyer par la suite du *spam*. Il s'est avéré que le *spam* de SIP est trois fois moins cher que le *spam* de téléphone de téléprospecteur. De plus, le *spam* de SIP peut être beaucoup plus perturbateur que le *spam* du *e-mail*.

6.5.2 SVSP et SIP

L'originalité de l'intégration de SVSP avec SIP est de mettre la sécurité de l'appel sur le canal *data* pour éviter de toucher à la signalisation de SIP. Le prototype décrit précédemment est implémenté sur le canal RTP qui transporte les conversations entre les usagers. La plate-forme JAIN-SIP [JAIN-SIP] utilisée se base sur Java qui permet une séparation entre l'API de l'application et le stack SIP. Les différentes classes Java relatives au prototype SVSP sont entièrement réutilisées dans l'intégration avec le standard de voix sur IP SIP. Cette intégration ne présente aucune modification de la signalisation de SIP pour l'établissement et la terminaison de l'appel SIP entre deux UAs. Le déploiement de SVSP s'effectue lors de l'établissement du canal RTP pour le

transport des médias. Les mêmes spécifications techniques utilisées et des résultats similaires des paramètres de performance sont obtenus comme suit :

- Spécifications techniques:
  - Réseau local: Ethernet à 100 Mbps ;
  - Longueur clé publique/prive: 1024 bits ;
  - Longueur clé symétrique: 64 bits ;
  - Signatures: DSA ;
  - MacTag: HMAC-SHA1 ;
  - Chiffrement symétrique: DES ;
  - Chiffrement asymétrique: RSA.

Table 9 – spécification technique

Machines	Processeur	Mémoire	Plateforme
Machine Client	Intel Centrino1.5GHZ	512 MOctets	JDK5.0,WINXP
Machine Serveur	Intel Pentium4 2.0GHZ	256 MOctets	JDK5.0,WINXP

- Mesures effectuées
  - Temps moyen mis pour commencer la session sécurisée(10 mesures): 502.8 ms ;
  - Temps moyen mis pour terminer la session sécurisée(10 mesures): 317.5 ms ;
  - Délai moyen introduit par le paquétiseur de voix (50 paquets): 0,432 ms ;
  - Délai moyen introduit par dépaquétiseur de voix (50 paquets): 3.6 ms ;
  - Débit du codec G.723 sans SVSP: 5 à 6 Kbps ;
  - Débit du codec G.723 avec SVSP: 10 à 11 Kbps ;
  - Overhead : (32(entêteSVSP+Mactag)/80(paquet total)): 40%.

6.6 Conclusion

Dans ce chapitre nous avons analysé les différents composants du protocole SVSP ainsi que ses limitations. Nous avons détaillé une implémentation d'un prototype de SVSP suivi d'une intégration avec le protocole SIP. Nous avons vu les vulnérabilités du protocole SIP et comment le protocole SVSP permet de les minimiser afin de pouvoir assurer la disponibilité des ressources capables de nous permettre d'effectuer des communications à bas prix. Cependant, afin de déterminer la robustesse du protocole SVSP, une analyse cryptographique est nécessaire. Notre perspective est de le valider formellement en utilisant l'outil AVISPA qui permettra de nous indiquer ses vulnérabilités face à des attaques bien connues comme l'analyse de trafic et les attaques de l'homme de milieu.

Chapitre 7

7 Conclusion générale et perspectives

7.1 Conclusion générale

Très peu de solutions globales de sécurité ont été proposées pour les réseaux de téléphonie. Chaque infrastructure apporte ses propres mécanismes de sécurité engendrant un appel non sécurisé de bout en bout et traversant différentes plates-formes de téléphonie. Le seul protocole offrant une sécurité de bout en bout à l'appel téléphonique est proposé par la NSA, et qui est le FNBDT, devenu SCIP. En revanche, ce protocole reste sous la coupe du pouvoir des Etats-Unis et des gouvernements membres du *NATO*. En plus, toutes les solutions existantes incluant FNBDT, malgré qu'elles ne soient pas toutes interopérables entre elles, n'offrent pas les majeurs services de sécurité nécessaires pour sécuriser les usagers et leurs conversations téléphoniques.

Ainsi, nous pouvons diviser notre travail dans cette thèse en trois grandes parties :

1. L'analyse des architectures de téléphonie et des solutions de sécurité existantes.
2. La définition d'une nouvelle architecture de sécurité pour la téléphonie.
3. La conception, l'étude d'intégration et l'implémentation d'un nouveau protocole de sécurité.

Dans la première partie (chapitre 2), nous avions pu exposer les grandes infrastructures de téléphonie, poser la problématique de la sécurisation de la téléphonie, définir les vulnérabilités de ces architectures, et enfin étudier et analyser les solutions de sécurité existantes à l'heure actuelle. Nous avions pour but de déterminer les exigences de la téléphonie en terme de sécurité et les lacunes dans les solutions existantes. Ceci nous a permis de comparer ces différentes architectures et leurs solutions de sécurité et de pouvoir arborer d'une part, leurs avantages et inconvénients et d'autre part, les exigences qu'ils ne peuvent pas satisfaire.

Dans la deuxième partie (chapitres 3 et 4), nous avons proposé une nouvelle architecture de sécurité qui s'adapte aux architectures de téléphonie actuelles et futures, ainsi qu'un protocole de sécurité qui matérialise cette architecture. Cette dernière prend en considération les spécificités de la téléphonie ainsi que les contraintes du flux de parole. L'architecture de sécurité (chapitre 3) offre un cadre sécuritaire à la téléphonie tout en tenant compte de ses caractéristiques et exigences pour éviter d'augmenter les délais susceptibles de pénaliser la qualité de la parole. Les objectifs de cette architecture sont d'assurer les services de sécurité à l'application téléphonique tout en assurant aussi la mobilité de l'usager. A l'heure actuelle, les certificats jouent un rôle prépondérant dans l'authentification des usagers et non pas uniquement des machines. Cependant, les certificats se basent sur des méthodes de chiffrement asymétrique nécessitant une puissance de calcul et des ressources assez importantes et exigeantes en terme de temps. Le chiffrement symétrique requiert moins de ressources et de temps pour chiffrer les

informations. Afin de répondre à ces différentes données, nous avons proposé deux méthodes pour assurer l'authentification mutuelle des usagers vis-à-vis de leurs correspondants. La première méthode prend en considération la présence d'un commutateur auquel l'usager est abonné, nous considèrerons alors l'utilisation d'une entité de confiance avec qui l'usager partagera une clé symétrique, nous la placerons au niveau du commutateur privé ou public auquel l'usager est raccordé. Cette entité partagera aussi avec l'utilisateur un identifiant unique capable de l'authentifier. Ces informations partagées seront protégées et sauvegardées sur une carte intelligente (*smart card*) qui en même temps assurera la mobilité de l'usager. Cette entité de confiance joue aussi le rôle d'intermédiaire pour transporter les messages nécessaires à l'authentification de l'usager distant L'authentification mutuelle des entités de confiance s'effectuera par des certificats. Cette architecture introduit aussi une nouvelle ouverture pour assurer la non répudiation des appels entre les usagers communicants par la livraison de rapports signés par les entités de confiance. La deuxième méthode consiste à utiliser les certificats directement avec les usagers sans avoir à passer par une entité tierce de confiance. Le chiffrement des données utiles (parole) échangées entre les usagers s'effectuera par un cryptage symétrique basé sur une clé partagée échangée entre les entités communicantes et autehntifiées, l'intégrité sera assurée par des fonctions de hachage unidirectionnelle et le non rejeu par l'application d'une fenêtre glissante appliquée aux numéros de séquence des informations échangées.

Afin de mettre en œuvre l'architecture de sécurité proposée, un protocole d'échange de messages est nécessaire pour l'application et le transport des différents mécanismes de sécurité utilisé et pour assurer une abstraction des réseaux sous-jacents. Ce protocole a pour rôle de fournir les paramètres propices à la sécurité et qui de sont pas assurer par les réseaux sous-jacents. Ce protocole que nous nommerons SVSP (pour Simple Voice Security Protocol) est décrit au chapitre 4 de ce mémoire de thèse.

Une étude de l'intégration de SVSP dans les infrastructures de téléphonie a été nécessaire afin de pouvoir assurer le placement des entités de confiance et d'analyser les limitations et les moyens disponibles pour le déploiement de SVSP au sein de chaque infrastructure de téléphonie. Le chapitre 5 propose l'étude de cette intégration dans trois plates-formes de téléphonie différentes, comme le réseau téléphonique commuté, le réseau mobile GSM et le réseau IP basé sur le standard H.323.

Cependant, l'architecture de sécurité et le protocole SVSP restent du domaine des propositions tant qu'une implémentation n'est pas effectuée pour spécifier leurs limitations et leurs avantages et donc la mise en œuvre de SVSP à échelle réelle. L'analyse du déploiement de SVSP et son prototype sont décrits dans le chapitre 6. Ainsi, nous verrons les limitations et les vulnérabilités liées à l'utilisation des cartes intelligentes ainsi que l'intégration de SVSP avec le standard SIP dans un environnement IP.

Toutefois, d'autres travaux sont nécessaires avant que SVSP soit réellement exploitable dans un environnement à fortes contraintes de sécurité. En effet, notre protocole n'est pas pour autant exempté de toute reproche et une validation formelle orientée sécurité avec l'outil d'analyse cryptographique AVISPA (ou autres) est

nécessaire pour connaître les vulnérabilités de SVSP aux attaques MitM et aux analyses de trafic. Nous pouvons citer à titre d'exemple que le protocole ne définit pas un mécanisme de sécurisation des bases de données UDB et RDB.

7.2 Perspectives et travaux futurs

Les perspectives de nos travaux sont surtout basées sur l'architecture de sécurité proposée, et plus spécifiquement sur le protocole SVSP. Notre intérêt porte sur les axes suivants :

- *Utilisation de SVSP pour les applications de voip sur des réseaux sans fils :*
  Un des problèmes spécifiques aux réseaux mobiles est la corrélation entre l'utilisateur nomade et le lieu où celle-ci a été transmise. Une comparaison de SVSP avec d'autres solutions d'authentification pour les environnements mobiles tels que le protocole EAP-TLS ou SEP [hajjeh] devrait être effectué.

- *Etendre SVSP pour sécuriser les applications de voip sur des réseaux P2P :*
  Une application voip qui se base sur une technologie *peer-to-peer* n'utilise pas un serveur central par lequel passe les appels, mais plutôt chaque client recherche d'autres clients d'une façon décentralisée. De ces connections, ce client construira alors un réseau qui l'aidera à rechercher d'autres. Aussi, les clients se connectent et se déconnectent du réseau d'une façon dynamique. Notre perspective est de pouvoir intégrer SVSP pour sécuriser les communications téléphoniques effectuées depuis un réseau P2P vers le réseau fixe pour assurer la sécurisation de bout en bout de l'appel.

- *Etendre SVSP à la communication de groupe :*
  SVSP offre une sécurisation des appels téléphoniques point à point. Pour les appels conférence ou les communications de groupe, SVSP offre avec la méthode qui utilise les entités tierces une solution pour authentifier les membres du groupe adhérant à la conférence. Notre perspective est de pousser nos recherches encore plus loin afin de pouvoir intégrer SVSP au service *multicast* du multimédia pour sécuriser une distribution large des données audio et/ou vidéo sur Internet.

- *SVSP - solution anti-spam :*
  Le grand public ne s'occupe pas de faire de la sécurité de la téléphonie. Il a tout juste besoin de pouvoir effectuer des appels téléphoniques au moindre coût avec une qualité de service suffisante pour avoir une conversation cohérente. Notre contribution ne se retreint pas uniquement à sécuriser les communications téléphoniques des industriels, des hauts fonctionnaires de l'état et des politiciens. Mais elle s'étend aussi pour couvrir les besoins du simple utilisateur lui permettant de téléphoner. La téléphonie migre de nos jours de plus en plus vers une application voix sur IP basée sur le protocole SIP. Cependant nous avons vu dans le chapitre précédent les attaques de *Spam* contre ce protocole qui affecte la disponibilité du service de téléphonie. Les méthodes d'authentification offertes par SVSP permettent d'être une solution idéale pour limiter les effets des attaques de *Spam*.

- Finalement, nous nous orientons à faire valoir et à pousser SVSP et le modèle associé à l'IETF.

Glossaire

3DES	Triple Data Encryption Standard
ADSL	Adaptative Digital Subscriber Line
AES	Advanced Encryption Standard
AH	Authentication Header
AKA	Authentication and Key Agreement
API	Application Protocol Interface
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
AuC	Authentication Center
BSC	Base Station Controller
BSS	Base Station Subsystem
BTS	Base Transceiver Station
CA	Certification Authority
CAS	Channel Associated Signaling
CDMA	Code Division Multiple Access
CLNP	ConnectionLess Network Protocol
COS	Card Operating System
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRL	Certificat Revocation List
cRTP	compressed Real-time Transport Protocol
DCCP	Datagram Congestion Control Protocol
DES	Data Encryption Standard
DH	Diffie-Hellam
DHCP	Dynamic Host Configuration Protocol
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
DUP	Data User Part
EAP	Extensible Authentication Protocol
EIR	Equipment Identity Register
ESP	Encapsulating Security Protocol
ETSI	European Telecommunication Standardization Institute
FA	Foreign Agent
FAI	Fournisseur d'Accès Internet
FNBDT	Future Narrowband Digital Terminal
FSM	Finite State Machine
GEA	GPRS Encryption Algorithm
GGSN	Gateway GPRS Support Node
GMSC	Gateway MSC
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GTAA	Global Trusted Authentication Authority
GTAA	Global Trusted Authentication Authority
GTP	GPRS Tunneling Protocol

HA	Home Agent
HLR	Home Location Register
HMAC-MD5	keyed-hash message authentication code – MD5
HMAC-SHA1	keyed-hash message authentication code – SHA1
HTTP	HyperText Transfer Protocol
HTTPS	HTTP with SSL
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IMAP	Internet Message Access Protocol
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IMT-2000	International Mobile Telecommunications-2000
IN	Intelligent Network
INAP	Intelligent Network Application Part
IP	Internet Protocol
IPsec	Internet Protocol Security
ISAKMP	Internet Security Association and Key Management Protocol
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part
IWF	InterWorking Function
JAIN-SIP	Java APIs for Intelligent Network – Session Initiation Protocol
MAC	Message Authentication Code
MAP	Mobile Application Part
MAP	Mobile Application Part
MCU	Multipoint Control Unit
MD5	Message-Digest algorithm 5
ME	Mobile Equipment
MELP	Mixed Excitation Linear Prediction
MGCP	Media Gateway Control Protocol
MIC	Modulations par Impulsion et Codages
MIKEY	Multimedia Internet KEYing
MitM	Man in the middle
MP	Multi-Processor
MS	Mobile Station
MSC	Mobile-services Switching Center
MTU	Maximum Transport Unit
NAT	Network Address Translation
NBSV-II	Narrow Band Secure Voice
NGN	Next Generation Network
NNI	Network to Network Interface
NSA	National Security Agency
NSS	Network Subsystem
NT1	Network Termination 1
OMC	Operation and Management Center
OSI	Open Systems Interconnect

OSS	Operation Support Subsystem
NATO	North Atlantic Treaty Organisation
PABX	Private Automatic Branch eXchange
PAN	Personal Area Network
PC	Personal Computer
PCM	Pulse Code Modulation
PDA	Personal digital assistants
PEAP	Protected Extensible Authentication Protocol
PIM	Protocol Independent Multicast
PKI	Public Key Infrastructure
PLMN	Public Land Mobile Network
PMK	Pair Wise Master Key
PMTU	Path Maximum Transport Unit
POP	Post Office Protocol
PRF	Pseudo Random Function
PSK	Pre Shared Key
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RC-4	Rivest Cipher 4 algorithm
RDB	Report DataBase
RDB	Report DataBase
RNC	Radio Network Controller
RNIS	Réseau Numérique à Intégration de Service
RNS	Radio Network Subsystems
RSA	Rivest-Shamir-Adleman algorithm
RSVP	Resource Reservation Protocol
RTC	Réseau téléphonique commuté
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
RTPC	Réseau Téléphonique Public Commuté
S/MIME	Secure / Multipurpose Internet Mail Extensions
SA	Security Association
SACK	Selective Acknowledge
SAML	Secure Authentication Markup Language
SAP	Session Announcement Protocol
SAPI	Secure Application Protocol Interface
SCCP	Signaling Connection and Control Part
SCP	Signaling Control Point
SDP	Session Description Protocol
SDP	Session Description Protocol
SDU	Service Data Unit
SGSN	Serving GPRS Support Node
SHA1	Secure Hash Algorithm
SIM	Subscriber Identity Module
SIP	Session Initiation Protocol

SLA	Service Level Agreement
SMS-GMSC	SMS-Gateway MSC
SN	Sequence Number
SRTP	Secure Real-time Transport Protocol
SS7	Signaling System 7
SSL	Secure Socket Layer
SSRC	Synchronization source
STP	Signaling Transfer Point
STU-II	Secure Telephone Unit – version II
STU-III	Secure Telephone Unit – version III
STUN	Simple Traversal of UDP through NAT
SVSP	Simple Voice Security Protocol
TAA	Trusted Authentication Authority
TAA	Trusted Authentication Authority
TCAP	Transaction Capability
TCP	Transport Control Protocol
TDMA	Time Division Multiple Access
TLS	Transport Layer Security
TLS-SIM	Transport Layer Security - Subscriber Identity Module
TMSI	Temporary Mobile Subscriber Identity
ToIP	Telephony over IP
TRAU	TRanscoder/Rate Adaptor Unit
TUP	Telephone User Part
TURN	Traversal Using Relay NAT
UAC	User Agent Client
UAS	User Agent Server
UDB	User DataBase
UDB	User DataBase
UDP	User Datagram Protocol
UE	User Equipment
UIT-T	Union International de Télécommunications – Secteur des Télécommunications
UMTS	Universal Mobile Telecommunications System
UNI	User to Network Interface
U-SIM	UMTS Subscriber Identity Module
UTRAN	UMTS Terrestrial Radio Access Network
VLAN	Virtual Local Area Network
VLR	Visitor Location Register
VoIP	Voice over IP
VPN	Virtual Private Network
W-CDMA	Wireless - Code-Division Multiple Access
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network

Bibliographie

[ITU-G114-03] ITU-T Recommendation G.114. One-way transmission time. Mai 2005.

[3DES] NIST, *"Triple Data Encryption Standard (3DES)"*, FIPS Publication 46-3, Octobre 1999.

[802.11idraft] IEEE Draft P802.11i/D6, "*Specification for Enhanced Security*", October 2003.

[802.1Xdraft] IEEE Draft P802.1X/D11, "*Standards for local and metropolitan area networks: Standard for port based network access control*", Mars 2001.

[AES] NIST, *"Advanced Encryption Standard (AES)"*, FIPS Publication 197, Novembre 2001 http://www.nist.gov/aes

[And04] S.Androutsellis-Theotokis et D.Spinellis. *"A survey of peer-to-peer content distribution technologies"*. ACM Computing Surveys, 36(4):335–371, December 2004.

[Aut02] T. Autret, L. Bellefin, M. Oble-Laffaire, *"Sécuriser ses échéances électroniques avec une PKI"*, Edition eyrolles, 2002, p.15.

[Bad04] M.Badra. *"Le transport et la sécurisation des échanges sur les réseaux sans fil"*,PhD thesis, ENST – Paris, 2004

[Bar02] R.Barbieri, D.Brushi, E.Rosti - *Voice over IPsec : Analysis and Solutions.* Univeristà degli Studi di Milano, 2002

[Bar05] C.Bardy, *" La ToIP prend d'assaut le grand public"* , Réseaux et Télécommunications, Vol.229, p.39, Avril 2005

[Bas04] S.A. Baset, H. Schulzrinne, *"An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol"*, Department of Computer Science, Columbia University, New York NY 10027, September 15, 2004

[Bav04] A.Bavosa, *"GPRS Security Threats and Solution Recommendations"*, Juniper Networks, June 2004

[Bil03] J. Bilien, E. Eliasson, J. Vatn. *Call Establishment Delay for Secure VoIP*, Diplom thesis, Stockholm, Sweden, 2003.

[Cam99] A. T. Campbell, et al. *"A Survey of Programmable Networks"*, ACM SIGCOMM Computer Communication Review, Vol. 29 No. 2 pg. 7-24, April 1999.

[Ced03] C. Llorens, L.Lévier, *"Tableaux de bord de sécurité réseaux "*, édition Eyrolles, octobre 2003.

[DCCPdraft] Kohler, E., Handley, M., Floyd, S., Padhye, J., "*Datagram Congestion Control Protocol*", expire septembre 2005

[DES] NIST, *"Data Encryption Standard (DES)"*, FIPS Publication 81, Octobre 1999.

[Dif76] W.Diffie, et M.E. Hellman, "New directions in cryptography", IEEE Transaction on Information Theory, 22, 6, p.654-655, 1976

[DTLSdraft] IETF Internet Draft. E.Rescorla et N.Modadugu. Datagram Transport Layer Security. Expire Décembre 2005.

[EAP-Double-TLSdraft] IETF Internet Draft. M., Badra, and P., Urien, "*EAP-Double-TLS Authentication Protocol*", draft-badra-eap-double-tls-03.txt, expire Avril 2006

[EAP-TTLSdraft] P. Funk, et S. Blake-Wilson, "*EAP Tunneled TLS Authentication Protocol (EAPTTLS)*", draft-ietf-pppext-eap-ttls-04.txt, expire Avril 2006

[FNBDT] General Dynamics, *"FNBDT Signaling Plan, Revision 1.1"*, National Security Agency, 01 Sept.1999

[IEEE 802.11] Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-1999, 1999.

[IEEE 802.11e] IEEE 802.11 WG. Draft Supplement to International Standard [for] Information Technology-Telecommunications and Information Exchange between systems LAN/MAN Specific Requirements. IEEE 802.11e/D2.o, Nov 2001.

[IMT-2000] IMT-2000, home page. http://www.itu.int/imt/

[ISDN] Security in ISDN – Chapitre 7 – http://csrc.nist.goc/publications/nistpubs/500-189/isdn7.ps

[ISO 17799] ANSI ISO 17799. Information technology - Security techniques - Code of practice for information security management. 2005.

[ITUT-H235-03] ITU-T Recommendation H.235. Security and encryption for H-Series multimedia terminals. Aout 2003.

[ITUT-H323-03] ITU-T Recommendation H.323. Packet-based multimedia communications systems, 2003

[Lag00] X.Lagrange, P.Godlewski, S.Tabbane, *"Réseaux GSM – Des principes à la norme*", Editions Hermes, 5ème Edition, 2000

[Lee04] H. Lee. "Securing IP Telephony: Secure SipFon", master thesis. Technical University of Hamburg-Harburg. Novembre 2004

[Luc00] Lucent Technologies, *"UMTS architecture"* Mars 2001

[Mil02] B.R., Miller, and B.A., Hamilton, "*Issues in Wireless Security WEP, WPA& 802.11*i", Proceedings of the 18th Annual Computer Security Applications Conference, Decembre 2002.

[Mod04] N.Modadugu, E.Rescorla. "The Design and Implementation of Datagram TLS". NSIS Working Group Meeting. IETF#61 (61st IETF - Washington, DC, USA, November 7-12, 2004)

[Mor02] R.Morrow, "Bluetooth Operation and Use", McGraw-Hill Telecom Professional, édition 2002.

[NSA] National Security Agency home page : http://www.nsa.org/.

[OpenSSL] OpenSSL. Home page. http://www.openssl.org

[Orange] Orange Bookbased . Home page http://www.dynamoo.com/orange/summary.htm

[PFL03] C. Pfleeger, S. Lawrence Pfleeger, *"Security in Computing*", Prentice-Hall, 2003.

[Phreak] Phone phreaking. Home page. http://telephonetribute.com/phonephreaking.html

[RC4] R.Rivest, *"Rivest Cipher 4*", RSA Security, 1987 ; http://www.rsasecurity.com

[RFC1191] IETF RFC 1191. Path MTU Discovery. Novembre 1990.

[RFC1321] IETF RFC 1321. The MD5 Message-Digest Algorithm, Avril 1992

[RFC1889] IETF RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. Janvier 1996

[RFC2104] IETF RFC 2104. HMAC: Keyed-Hashing for Message Authentication. Février 1997.

[RFC2246] IETF RFC 2246. The TLS Protocol Version 1.0. Janvier 1999

[RFC2284] IETF RFC 2284. PPP Extensible Authentication Protocol. Mars 1998.

[RFC2401] IETF RFC 2401. Security Achitecture for the Internet Protocol. Novembre 1998.

[RFC2408] IETF RFC 2408. Internet Security Association and Key Management Protocol (ISAKMP). Novembre 1998.

[RFC3261] IETF RFC 2543. SIP: Session Initiation Protocol. Juin 2002.

[RFC2633] IETF RFC 2633. S/MIME Version 3 Message Specification. June 1999.

[RFC2705] IETF RFC 2705 - Media Gateway Control Protocol (MGCP) Version 1.0, octobre 1999

[RFC2716] IETF RFC 2716. PPP EAP TLS Authentication Protocol. Octobre 1999.

[RFC3303] IETF RFC 3303. Middlebox communication architecture and framework. Aout 2002.

[RFC3489] IETF RFC 3489. Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). Mars 2003.

[RFC3545] IETF RFC 3545 - Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering. Juillet 2003

[RFC3711] IETF RFC 3711. The Secure Real-time Transport Protocol. Mars 2004.

[RFC3830] IETF RFC 3830 - MIKEY: Multimedia Internet KEYing. Aout 2004

[Rig98] C. Rigault, *"Principes de commutation numérique"*, Hermes, Collection pédagogique de télécommunications, Paris, 1998.

[RSA] R. Rivest, A. Shamir, L. Adleman, *"RSA"*, ISO/IEC 9796, 1977.

[Sal75] J.Saltzer et M. Schroeder, *"The Protection of Information in Computer Systems"*, *Proceedings of the IEEE*, **Vol.63**, **No.9** (September 1975), p.1278.

[SAMLdraft] IETF Internet draft. H. Tschofenig, et al. Using SAML for SIP. Expire Janvier 2006.

[Scout] Internet Scout Project. Home page. http://scout.wisc.edu/

[Setec] Setec a Gemplus company. Home page. http://www.setec.com/

[Seu03] E. Seurre, P.Savelli et J-P.Pietri, *"GPRS for Mobile Internet"*, Edition Artech House, 2003

[SHA93] NIST, *"Secure Hash Standard (SHA-1)"*, FIPS PUB 180-1,May 1993

[Shar02] M.Sharif, et al., *"Providing Secure Communication Services on the Public Telephone Network Infrastructures"*, 2002 www.cs.nps.navy.mil/people/faculty/bmichael/pubs/

[Sigsaly] IEEE Spectrum September 1984, pages 70-80.]

http://www.nsa.gov/publications/publi00020.cfm

[Sim05] L. Simson. Garfinkel, *"VoIP and Skype Security"*, Skype Security Overview – Rev 1.5 – Mars 2005

[STU-III] Department of Defense Security Institute, *"STU-III Handbook for Industry"*, http://www.tscm.com/STUIIIhandbook.html, February 1997

[Ten97] D. Tennenhouse, et al. *"A Survey of Active Network Research"*. IEEE Communications, January 1997.

[Thi97] J-P.Tran Thiet, *" Télécommunications ! nouvelle donne, nouveaux enjeux pour votre entreprise"* , Les Editions d'Organisation, 1997.

[TURNdraft] IETF Internet draft. J. Rosenberg, R. Mahy, C. Huitema. *"TURN: traversal using relay NAT"*. Internet draft, Internet Engineering Task Force, expire Avril 2006

[Wil99] K.G.Wilson, *"Du monopole à la compétition : la déréglementation des télécommunications au Canada et aux Etats-Unis"* , Sainte-Foy, QC : Télé-université, 1999. 535p. ISBN 276241234X

[Wus02] E.Wustenhoff, *"Service Level Agreement in the Data Center"*, Sun Professional Services Sun BluePrints™ OnLine - April 2002

[JavaCardSecurity] Attaques sur les cartes intelligentes Java. Disponible au site http://www.securingjava.com/chapter-eight/chapter-eight-5.html

[McG98] G. McGraw, Smart cards, "Java cards and security", Reliable Software Technologies, January 1998

[Sch96] B. Schneier. *Applied Cryptography*, 2ème Edition, John Wiley & Sons, New York, 1996.

Annexe A : Publications associées à cette thèse

C.Bassil, et al.., “Etat de l’art de la sécurité de la Voix “, 3ème conférence SAR (Sécurité et Architecture des Réseaux), Juin 2004, La Londe Les Maures- France.

C.Bassil, A.Serhrouchni, N.Rouhana, “Critical voice network security analysis and new approach for securing Voice over IP Communications”, 3ème conference SETIT (Sciences Electroniques, Technologies de l'Information et des Télécommunications), Mars 2005, Sousse - Tunisie

C.Bassil, A.Serhrouchni, N.Rouhana, “Toward a new perspective of Secure Real-Time Transport Protocol with Active network”, 3ème conférence SETIT (Sciences Electroniques, Technologies de l'Information et des Télécommunications), Mars 2005, Sousse - Tunisie

C.Bassil, A.Serhrouchni, N.Rouhana, “Critical analysis and new perspective for securing voice networks”, 4ème conférence ICN (International Conference on Networking), Avril 2005, Ile de la Réunion

C.Bassil, A.Serhrouchni, N.Rouhana, “Towards securing VoIP communications with FNBDT”, 12ème conférence ICT (International Conference on Telecommunications), Mai 2005, CapeTown – Afrique du Sud

C.Bassil, A.Serhrouchni, N.Rouhana, “Towards a new Security Architecture for Telephony”, 5ème conférence ICN (International Conference on Networking), Avril 2006, Ile Maurice

Annexe B : code source du prototype SVSP

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;

public class UserAgentClient
{
	static final int port = 10000;
	static public void main (String[] args) throws Exception
	{
		Socket client=new Socket("127.0.0.1",port);
		BufferedInputStream entre =new BufferedInputStream (client.getInputStream());
		PrintStream sortie=new PrintStream (client.getOutputStream());
		System.out.println("Sending U_RUCAreq ...");
		Date start=new Date();
		String start_time=start.toString();
		SendU_RUCAreq request=new SendU_RUCAreq();
		SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
		byte[] randomA=new byte[64];
		sr.nextBytes(randomA);
		byte[] SID =new byte[1];
		sr.nextBytes(SID);
		byte[] sequenceNumber=new byte[2];
		sr.nextBytes(sequenceNumber);
		byte[]
		mess=request.buildMessage("tonycert.cer","tonykey",randomA,SID,sequenceNumber);

		sortie.write(mess,0,mess.length);
		System.out.println("Receiving U_RUCAreq ...");
		mess=new byte[902];
		int i=entre.read(mess,0,mess.length);
		//System.out.println(new String(mess));
		ReceiveU_RUCA receive1=new ReceiveU_RUCA();
		byte[] receivedmess=receive1.getRandom("carolecert.cer",mess,596);
		byte[] randomB=receivedmess;
		//System.out.println(new String(randomB));
		System.out.println("Sending U_RUCArep ...");
		randomA[0]=(byte)128;
		BigInteger p = new BigInteger(1,randomA);
		BigInteger g = new BigInteger(1,randomB);
		int exp =511;

		//System.out.println(randomA.length);
		//System.out.println(new String(randomA));
		//System.out.println(new String(randomB));
		//System.out.println(p+";"+g+";"+exp);
		KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
		DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
		keyGen.initialize(dhSpec);
		KeyPair keypair = keyGen.generateKeyPair();
```

```
                    // Get the generated public and private keys
                    PublicKey DHpublicKey = keypair.getPublic();
                    PrivateKey DHprivateKey = keypair.getPrivate();
                    //System.out.println(DHpublicKey.getFormat());
                    sequenceNumber[0]++;
                    if(sequenceNumber[0]==(byte)0)
                    sequenceNumber[1]++;
                    SendU_RUCArep request2=new SendU_RUCArep();

          mess=request2.buildMessage("12345678","tonykey","carolecert.cer",SID,sequenceNumber,rando
mB,DHpublicKey);
                    //System.out.println(mess.length);
                    sortie.write(mess,0,mess.length);
                    System.out.println("Receiving U_RUCArep ...");
                    while(entre.available()==0);
                    mess=new byte[entre.available()];
                    i=entre.read(mess,0,mess.length);
                    ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
                    receivedmess=receive2.getKey("carolecert.cer","tonykey",randomA,mess);
                    X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
                    KeyFactory keyFact = KeyFactory.getInstance("DH");
                    DHpublicKey = keyFact.generatePublic(x509KeySpec);
                    KeyAgreement ka = KeyAgreement.getInstance("DH");
                    ka.init(DHprivateKey);
                    ka.doPhase(DHpublicKey, true);
                    String algorithm = "DES";

     // Generate the secret key
     SecretKey secretKey1 = ka.generateSecret(algorithm);
     System.out.println("Sending U_RUCAconf ...");
     sequenceNumber[0]++;
     if(sequenceNumber[0]==(byte)0)
     sequenceNumber[1]++;
     SendU_RUCAconf request3=new SendU_RUCAconf();
                    mess=request3.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
                    sortie.write(mess,0,mess.length);
                    System.out.println("Receiving U_RUCAconf...");
                    mess=new byte[141];
                    i=entre.read(mess,0,mess.length);
                    ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
                    receivedmess=receive3.getToken(mess,secretKey1);
                    byte[] TokenB=receivedmess;
                    System.out.println("Setup completed!");
                    SendMessage sm=new
SendMessage(secretKey1,"client\\clearoutputlog.txt","client\\cipheroutputlog.txt");
      ReceiveMessage rm=new
ReceiveMessage(secretKey1,"client\\clearinputlog.txt","client\\cipherinputlog.txt");
          rm.start();
     sm.start();

          Endcallpanel f= new Endcallpanel();
          f.show();
     while(f.isShowing());
     System.out.println("Call ended!");
     sm.stop();
```

```
rm.stop();
byte[] endcallmess="endcall".getBytes();
InetAddress adresse=InetAddress.getByName("127.0.0.1");
DatagramPacket envoi=new DatagramPacket(endcallmess,endcallmess.length,adresse, 6000);
DatagramSocket socket = new DatagramSocket();
                socket.send(envoi);
                socket.close();
                System.out.println("Sending U_ESMSreq...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSreq request4=new SendU_ESMSreq();
                mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSreq...");
                while(entre.available()==0);
                mess=new byte[36];
                i=entre.read(mess,0,mess.length);
                ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
                receive4.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSrep...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSrep request5=new SendU_ESMSrep();
                mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSrep...");
                mess=new byte[36];
                i=entre.read(mess,0,mess.length);
                ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
                receive5.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSind...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSind request6=new SendU_ESMSind();
                mess=request6.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSind...");
                mess=new byte[109];
                i=entre.read(mess,0,mess.length);
                ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
                receivedmess=receive6.getToken(mess,secretKey1);
                String rec=new String(receivedmess);
                System.out.println("Returned Tokon:"+rec);
                System.out.println("Sending U_ESMSconf...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSconf request7=new SendU_ESMSconf();
                Date end=new Date();
                String end_time=end.toString();

        mess=request7.buildMessage("12345678","87654321",start_time,end_time,"tonykey",SID,sequen
ceNumber,secretKey1);
```

```
                    sortie.write(mess,0,mess.length);
                    System.out.println("Receiving U_ESMSconf...");
                    while(entre.available()==0);
                    mess=new byte[entre.available()];
                    i=entre.read(mess,0,mess.length);
                    ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
                    receivedmess=receive7.getReport("carolecert.cer",mess,secretKey1);
                    FileOutputStream fos=new FileOutputStream("client\\clientreportlog.txt");
                    fos.write(receivedmess);
                    fos.close();
                    rec=new String(receivedmess);
                    System.out.println("Distante user report:"+rec);
        client.close();
        System.exit(0);

            }
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;

public class UserAgentServer
{
        static final int port = 10000;
        static public void main (String[] args) throws Exception
        {
                ServerSocket serveur=new ServerSocket(port);
                System.out.println("Waiting for connection...");
                Socket socket=serveur.accept();
                serveur(socket);
        }
        static void serveur(Socket socket) throws Exception
        {
                BufferedInputStream entre =new BufferedInputStream (socket.getInputStream());
                PrintStream sortie=new PrintStream (socket.getOutputStream());
                System.out.println("Client connected!");
                System.out.println(" waiting for U_RUCAreq ... ");
                byte[] mess=new byte[926];
                int i=entre.read(mess,0,926);

                ReceiveU_RUCA receive1=new ReceiveU_RUCA();
                byte[] receivedmess=receive1.getRandom("tonycert.cer",mess,620);
                //System.out.println(receivedmess.length);
                System.out.println("Sending U_RUCAreq...");
                byte[] randomA=receivedmess;
                Date start=new Date();
                String start_time=start.toString();
                SendU_RUCAreq request=new SendU_RUCAreq();
                SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
                byte[] randomB=new byte[64];
                sr.nextBytes(randomB);
                byte[] SID =new byte[1];
                sr.nextBytes(SID);
                byte[] sequenceNumber=new byte[2];
                sr.nextBytes(sequenceNumber);

        mess=request.buildMessage("carolecert.cer","carolekey",randomB,SID,sequenceNumber);
                //System.out.println(mess.length);
                randomA[0]=(byte)128;
                BigInteger p = new BigInteger(1,randomA);
        BigInteger g = new BigInteger(1,randomB);
        int exp =511;
        //System.out.println(new String(randomA));
        //System.out.println(new String(randomB));
        //System.out.println(p+";"+g+";"+exp);
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
    DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
    keyGen.initialize(dhSpec);
```

```
KeyPair keypair = keyGen.generateKeyPair();
// Get the generated public and private keys
PublicKey DHpublicKey = keypair.getPublic();
PrivateKey DHprivateKey = keypair.getPrivate();
sortie.write(mess,0,mess.length);
System.out.println("Receiving U_RUCA.rep ...." );
entre =new BufferedInputStream (socket.getInputStream());
while(entre.available()==0);
            mess=new byte[entre.available()];
            //System.out.println(mess.length);
            i=entre.read(mess,0,mess.length);
            ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
            receivedmess=receive2.getKey("tonycert.cer","carolekey",randomB,mess);
            X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
KeyFactory keyFact = KeyFactory.getInstance("DH");
PublicKey publicKey = keyFact.generatePublic(x509KeySpec);
//System.out.println(publicKey.getFormat());
KeyAgreement ka = KeyAgreement.getInstance("DH");
ka.init(DHprivateKey);
ka.doPhase(publicKey, true);
String algorithm = "DES";

// Generate the secret key
SecretKey secretKey1 = ka.generateSecret(algorithm);
System.out.println("Sending U_RUCArep ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCArep request2=new SendU_RUCArep();

    mess=request2.buildMessage("87654321","carolekey","tonycert.cer",SID,sequenceNumber,rando
mA,DHpublicKey);
            sortie.write(mess,0,mess.length);
            System.out.println("Receiving U_RUCAconf ...");
            mess=new byte[141];
            i=entre.read(mess,0,141);
            ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
            receivedmess=receive3.getToken(mess,secretKey1);
            byte[] TokenB=receivedmess;
            System.out.println("Sending U_RUCAconf ...");
            sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCAconf request3=new SendU_RUCAconf();
            mess=request3.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
            sortie.write(mess,0,mess.length);
            System.out.println("Setup completed!");
            SendMessage1 sm=new
SendMessage1(secretKey1,"server\\clearoutputlog.txt","server\\cipheroutputlog.txt");
ReceiveMessage1 rm=new
ReceiveMessage1(secretKey1,"server\\clearinputlog.txt","server\\cipherinputlog.txt");
    rm.start();
sm.start();
            boolean endcall=false;
            byte[] endcallmess=new byte["endcall".getBytes().length];
            DatagramPacket packet=new DatagramPacket(endcallmess,endcallmess.length);
```

```
                DatagramSocket udpsocket = new DatagramSocket(6000);
                do
                {
                        udpsocket.receive(packet);
                        if(new String(endcallmess)!="")
                        endcall=true;
                }
                while(endcall==false);
                udpsocket.close();
                sm.stop();
                rm.stop();
                System.out.println("Distante user ended the call!");
                System.out.println("Receiving U_ESMSreq ...");
                mess=new byte[36];
                i=entre.read(mess,0,36);
                ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
                receive4.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSreq ...");
                sequenceNumber[0]++;
        if(sequenceNumber[0]==(byte)0)
        sequenceNumber[1]++;
        SendU_ESMSreq request4=new SendU_ESMSreq();
                mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSrep ...");
                mess=new byte[36];
                i=entre.read(mess,0,36);
                ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
                receive5.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSrep ...");
                sequenceNumber[0]++;
        if(sequenceNumber[0]==(byte)0)
        sequenceNumber[1]++;
        SendU_ESMSrep request5=new SendU_ESMSrep();
                mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSind ...");
                mess=new byte[109];
                i=entre.read(mess);
                ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
                receivedmess=receive6.getToken(mess,secretKey1);
                String rec=new String(receivedmess);
                System.out.println("Returned Tokon:"+rec);
                System.out.println("Sending U_ESMSind ...");
                sequenceNumber[0]++;
        if(sequenceNumber[0]==(byte)0)
        sequenceNumber[1]++;
        SendU_ESMSind request6=new SendU_ESMSind();
                mess=request6.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
                sortie.write(mess,0,mess.length);
                System.out.println("Receiving U_ESMSconf ...");
                while(entre.available()==0);
                mess=new byte[entre.available()];
                i=entre.read(mess,0,mess.length);
                ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
                receivedmess=receive7.getReport("tonycert.cer",mess,secretKey1);
```

```
                    FileOutputStream fos=new FileOutputStream("server\\serverreportlog.txt");
                    fos.write(receivedmess);
                    fos.close();
                    rec=new String(receivedmess);
                    System.out.println("Distante user report:"+rec);
                    System.out.println("Sending U_ESMSconf ...");
                    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSconf request7=new SendU_ESMSconf();
                    Date end=new Date();
                    String end_time=end.toString();

          mess=request7.buildMessage("87654321","12345678",start_time,end_time,"carolekey",SID,seque
nceNumber,secretKey1);
                    sortie.write(mess,0,mess.length);
                    socket.close();
                    System.exit(0);


          }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_RUCAreq
{
	public byte[] buildMessage(String cername,String key ,byte[] rnd,byte[] SID,byte[]
sequenceNumber) throws Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=0;
		byte extlow1=8;
		FileInputStream fis=new FileInputStream(cername);
		CertificateFactory cf=CertificateFactory.getInstance("X.509");
		Collection c=cf.generateCertificates(fis);
		Iterator i=c.iterator();
		java.security.cert.Certificate cert = (java.security.cert.Certificate)i.next();

		byte[] certstream=cert.getEncoded();
		//System.out.println("cert"+certstream.length);
		int HEdatalength=certstream.length;
		HEdatalength=HEdatalength+4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		byte extlow2=16;
		byte[] usercap="AES;HMACwithSHA1;DiffieHellman".getBytes();
		//System.out.println("usercap"+usercap.length);
		int HEdatalength2=usercap.length;
		HEdatalength2=HEdatalength2+4;
		Integer HEdatalen2=HEdatalength2;
		byte HEdatalow2=HEdatalen2.byteValue();
		HEdatalen2=HEdatalen2.rotateRight(HEdatalen2,8);
		byte HEdatahigh2=HEdatalen2.byteValue();

		KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
		char[] password ={'p','a','s','s','w','o','r','d'};
		java.io.FileInputStream fis1 =
	new java.io.FileInputStream(".keystore");
		ks.load(fis1, password);
	fis.close();
	fis1.close();
```

```
                // get my private key
                char[]     keypassword ={'1','2','3','4','5','6'};
                PrivateKey myPrivateKey = (PrivateKey)ks.getKey(key, keypassword);
                Cipher   rsaCipher = Cipher.getInstance("RSA");
                // Initialize the cipher for encryption
        rsaCipher.init(Cipher.ENCRYPT_MODE, myPrivateKey);
        byte[] cipherRnd = rsaCipher.doFinal(rnd);
        //System.out.println(cipherRnd.length);


                Integer messagesize=12+HEdatalength+HEdatalength2+cipherRnd.length;
                byte messagelenlow=messagesize.byteValue();
                messagesize=messagesize.rotateRight(messagesize,8);
                byte messagelenhigh=messagesize.byteValue();
                byte[] entete=new byte[8];
                entete[0]=firstByte;
                entete[1]=SID[0];
                entete[2]=messagelenhigh;
                entete[3]=messagelenlow;
                entete[4]=sequenceNumber[0];
                entete[5]=sequenceNumber[1];
                entete[6]=timestamphigh;
                entete[7]=timestamplow;

                byte[] HE1=new byte[HEdatalength];
                HE1[0]=exthigh;
                HE1[1]=extlow1;
                HE1[2]=HEdatahigh;
                HE1[3]=HEdatalow;
                System.arraycopy(certstream,0,HE1,4,certstream.length);

                byte[] HE2=new byte[HEdatalength2];
                HE2[0]=exthigh;
                HE2[1]=extlow2;
                HE2[2]=HEdatahigh2;
                HE2[3]=HEdatalow2;
                System.arraycopy(usercap,0,HE2,4,usercap.length);
                messagesize=8+HEdatalength+HEdatalength2+cipherRnd.length;

                byte[] svspmessage=new byte[messagesize];
                System.arraycopy(entete,0,svspmessage,0,entete.length);
                System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
                System.arraycopy(HE2,0,svspmessage,(entete.length+HE1.length),HE2.length);

        System.arraycopy(cipherRnd,0,svspmessage,(entete.length+HE1.length+HE2.length),cipherRnd.le
ngth);

                CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                byte[] buf=new byte[32];
                while(cis.read(buf)>0){}
                Long checksum=cis.getChecksum().getValue();
                byte bb1=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb2=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
```

```
                    byte bb3=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb4=checksum.byteValue();
                    byte[] bb={bb1,bb2,bb3,bb4};

                    byte[] entete2=new byte[12];
                    System.arraycopy(entete,0,entete2,0,entete.length);
                    System.arraycopy(bb,0,entete2,8,4);

                    messagesize=12+HEdatalength+HEdatalength2+cipherRnd.length;

                    svspmessage=new byte[messagesize];
                    System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                    System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);
                    System.arraycopy(HE2,0,svspmessage,(entete2.length+HE1.length),HE2.length);

          System.arraycopy(cipherRnd,0,svspmessage,(entete2.length+HE1.length+HE2.length),cipherRnd.
length);


                    Signature rsa=Signature.getInstance("MD5withRSA");
                    rsa.initSign(myPrivateKey);
                    rsa.update(svspmessage);
                    byte[] sig=rsa.sign();
          //        System.out.println("signature"+sig.length);
                    byte[] message=new byte[(sig.length+messagesize)];
                    System.arraycopy(svspmessage,0,message,0,messagesize);
                    System.arraycopy(sig,0,message,messagesize,sig.length);
                    return (message);

          }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import javax.crypto.spec.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_RUCArep
{
	public byte[] buildMessage(String ID,String key,String cername,byte[] SID,byte[]
sequenceNumber,byte[]rnd,PublicKey DHpubkey) throws Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=32;
		byte extlo w1=8;
		byte[] IDtoByte=ID.getBytes();
		byte[] rnd_id=new byte[(rnd.length+IDtoByte.length)];
		System.arraycopy(rnd,0,rnd_id,0,rnd.length);
		System.arraycopy(IDtoByte,0,rnd_id,rnd.length,IDtoByte.length);
		MessageDigest md = MessageDigest.getInstance("SHA1");
		md.update(rnd_id);
		byte[] hres = md.digest();
		//System.out.println("Hres"+hres.length);
		int HEdatalength=hres.length;
		HEdatalength=HEdatalength+4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		byte extlow2=16;
		byte[] DHkey=DHpubkey.getEncoded();
		//System.out.println("dhkey"+DHkey.length);
		int HEdatalength2=DHkey.length;
		HEdatalength2=HEdatalength2+4;
		Integer HEdatalen2=HEdatalength2;
		byte HEdatalow2=HEdatalen2.byteValue();
		HEdatalen2=HEdatalen2.rotateRight(HEdatalen2,8);
		byte HEdatahigh2=HEdatalen2.byteValue();
		FileInputStream fis=new FileInputStream(cername);
		CertificateFactory cf=CertificateFactory.getInstance("X.509");
		Collection c=cf.generateCertificates(fis);
		Iterator i=c.iterator();
		java.security.cert.Certificate cert = (java.security.cert.Certificate)i.next();
		PublicKey pubB=cert.getPublicKey();
		fis.close();
```

```
                    KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
                    char[] password ={'p','a','s','s','w','o','r','d'};
                    java.io.FileInputStream fis1 =
      new java.io.FileInputStream(".keystore");
                    ks.load(fis1, password);
          fis1.close();
          char[]    keypassword ={'1','2','3','4','5','6'};
                    PrivateKey myPrivateKey = (PrivateKey)ks.getKey(key, keypassword);
                    Cipher   rsaCipher = Cipher.getInstance("RSA");
                    // Initialize the cipher for encryption
          rsaCipher.init(Cipher.ENCRYPT_MODE, pubB);
          byte[] cipherID = rsaCipher.doFinal(IDtoByte);
          //System.out.println("cipherid"+cipherID.length);
          Integer messagesize=12+HEdatalength+HEdatalength2+cipherID.length;
                    byte messagelenlow=messagesize.byteValue();
                    messagesize=messagesize.rotateRight(messagesize,8);
                    byte messagelenhigh=messagesize.byteValue();
                    byte[] entete=new byte[8];
                    entete[0]=firstByte;
                    entete[1]=SID[0];
                    entete[2]=messagelenhigh;
                    entete[3]=messagelenlow;
                    entete[4]=sequenceNumber[0];
                    entete[5]=sequenceNumber[1];
                    entete[6]=timestamphigh;
                    entete[7]=timestamplow;

                    byte[] HE1=new byte[HEdatalength];
                    HE1[0]=exthigh;
                    HE1[1]=extlow1;
                    HE1[2]=HEdatahigh;
                    HE1[3]=HEdatalow;
                    System.arraycopy(hres,0,HE1,4,hres.length);

                    byte[] HE2=new byte[HEdatalength2];
                    HE2[0]=exthigh;
                    HE2[1]=extlow2;
                    HE2[2]=HEdatahigh2;
                    HE2[3]=HEdatalow2;
                    System.arraycopy(DHkey,0,HE2,4,DHkey.length);
                    messagesize=8+HEdatalength+HEdatalength2+cipherID.length;
                    byte[] svspmessage=new byte[messagesize];
                    System.arraycopy(entete,0,svspmessage,0,entete.length);
                    System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
                    System.arraycopy(HE2,0,svspmessage,(entete.length+HE1.length),HE2.length);

          System.arraycopy(cipherID,0,svspmessage,(entete.length+HE1.length+HE2.length),cipherID.leng
th);

                    CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                    byte[] buf=new byte[32];
                    while(cis.read(buf)>0){}
                    Long checksum=cis.getChecksum().getValue();
                    byte bb1=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
```

```
                    byte bb2=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb3=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb4=checksum.byteValue();
                    byte[] bb={bb1,bb2,bb3,bb4};

                    byte[] entete2=new byte[12];
                    System.arraycopy(entete,0,entete2,0,entete.length);
                    System.arraycopy(bb,0,entete2,8,4);

                    messagesize=12+HEdatalength+HEdatalength2+cipherID.length;
                    svspmessage=new byte[messagesize];
                    System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                    System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);
                    System.arraycopy(HE2,0,svspmessage,(entete2.length+HE1.length),HE2.length);

          System.arraycopy(cipherID,0,svspmessage,(entete2.length+HE1.length+HE2.length),cipherID.len
gth);

                    Signature rsa=Signature.getInstance("MD5withRSA");
                    rsa.initSign(myPrivateKey);
                    rsa.update(svspmessage);
                    byte[] sig=rsa.sign();
                    //System.out.println("signature"+sig.length);
                    byte[] message=new byte[(sig.length+messagesize)];
                    System.arraycopy(svspmessage,0,message,0,messagesize);
                    System.arraycopy(sig,0,message,messagesize,sig.length);
                    return (message);



}          }
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_RUCAconf
{
	public byte[] buildMessage(String ID,byte[] rnd,byte[] SID,byte[] sequenceNumber,SecretKey
secretkey ) throws Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=48;
		byte extlow1=0;
		byte[] IDtoByte=ID.getBytes();
		byte[] conf="Cverify=ok;Encryption=AES".getBytes();

		byte[]confmess=new byte[IDtoByte.length+conf.length];
		System.arraycopy(IDtoByte,0,confmess,0,IDtoByte.length);
		System.arraycopy(conf,0,confmess,IDtoByte.length,conf.length);
		//System.out.println("conf"+confmess.length);
		int HEdatalength=confmess.length;
		HEdatalength=HEdatalength+4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		byte[] token=new byte[IDtoByte.length+rnd.length];
	//	System.out.println("token"+token.length);
		System.arraycopy(IDtoByte,0,token,0,IDtoByte.length);
		System.arraycopy(rnd,0,token,IDtoByte.length,rnd.length);
		Integer messagesize=12+HEdatalength+token.length;
		byte messagelenlow=messagesize.byteValue();
		messagesize=messagesize.rotateRight(messagesize,8);
		byte messagelenhigh=messagesize.byteValue();
		byte[] entete=new byte[8];
		entete[0]=firstByte;
		entete[1]=SID[0];
		entete[2]=messagelenhigh;
		entete[3]=messagelenlow;
		entete[4]=sequenceNumber[0];
		entete[5]=sequenceNumber[1];
		entete[6]=timestamphigh;
		entete[7]=timestamplow;
```

```
                    byte[] HE1=new byte[HEdatalength];
                    HE1[0]=exthigh;
                    HE1[1]=extlow1;
                    HE1[2]=HEdatahigh;
                    HE1[3]=HEdatalow;
                    System.arraycopy(confmess,0,HE1,4,confmess.length);
                    messagesize=8+HEdatalength+token.length;
                    byte[] svspmessage=new byte[messagesize];
                    System.arraycopy(entete,0,svspmessage,0,entete.length);
                    System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
                    System.arraycopy(token,0,svspmessage,(entete.length+HE1.length),token.length);

                    CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                    byte[] buf=new byte[32];
                    while(cis.read(buf)>0){}
                    Long checksum=cis.getChecksum().getValue();
                    byte bb1=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb2=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb3=checksum.byteValue();
                    checksum=checksum.rotateRight(checksum,8);
                    byte bb4=checksum.byteValue();
                    byte[] bb={bb1,bb2,bb3,bb4};

                    byte[] entete2=new byte[12];
                    System.arraycopy(entete,0,entete2,0,entete.length);
                    System.arraycopy(bb,0,entete2,8,4);

                    messagesize=12+HEdatalength+token.length;

                    svspmessage=new byte[messagesize];
                    System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                    System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);
                    System.arraycopy(token,0,svspmessage,(entete2.length+HE1.length),token.length);

                    Mac mac=Mac.getInstance("HmacSHA1");
                    mac.init(secretkey);
                    byte[]mactag=mac.doFinal(svspmessage);
                    //System.out.println("mactag"+mactag.length);
                    byte[] message=new byte[(mactag.length+messagesize)];
                    System.arraycopy(svspmessage,0,message,0,messagesize);
                    System.arraycopy(mactag,0,message,messagesize,mactag.length);
                    return (message);


          }
}
```

```
import java.io.*;
import java.security.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_ESMSreq
{
	public byte[] buildMessage(byte[] SID,byte[] sequenceNumber, SecretKey secretkey) throws
Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=64;
		byte extlow1=0;
		int HEdatalength=4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		Integer messagesize=16;
		byte messagelenlow=messagesize.byteValue();
		messagesize=messagesize.rotateRight(messagesize,8);
		byte messagelenhigh=messagesize.byteValue();
			byte[] entete=new byte[8];
		entete[0]=firstByte;
		entete[1]=SID[0];
		entete[2]=messagelenhigh;
		entete[3]=messagelenlow;
		entete[4]=sequenceNumber[0];
		entete[5]=sequenceNumber[1];
		entete[6]=timestamphigh;
		entete[7]=timestamplow;

		byte[] HE1=new byte[HEdatalength];
		HE1[0]=exthigh;
		HE1[1]=extlow1;
		HE1[2]=HEdatahigh;
		HE1[3]=HEdatalow;
		messagesize=12;
		byte[] svspmessage=new byte[messagesize];
		System.arraycopy(entete,0,svspmessage,0,entete.length);
		System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
```

```
                Long checksum=cis.getChecksum().getValue();
                byte bb1=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb2=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb3=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb4=checksum.byteValue();
                byte[] bb={bb1,bb2,bb3,bb4};
                byte[] entete2=new byte[12];
                System.arraycopy(entete,0,entete2,0,entete.length);
                System.arraycopy(bb,0,entete2,8,4);
                messagesize=16;
                svspmessage=new byte[messagesize];
                System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);

                Mac mac=Mac.getInstance("HmacSHA1");
                mac.init(secretkey);
                byte[]mactag=mac.doFinal(svspmessage);
                //System.out.println("mactag"+mactag.length);
                byte[] message=new byte[(mactag.length+messagesize)];
                System.arraycopy(svspmessage,0,message,0,messagesize);
                System.arraycopy(mactag,0,message,messagesize,mactag.length);
                return (message);
        }
}
```

```
import java.io.*;
import java.security.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_ESMSrep
{
	public byte[] buildMessage(byte[] SID,byte[] sequenceNumber, SecretKey secretkey) throws
Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=80;
		byte extlow1=0;
		int HEdatalength=4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		Integer messagesize=16;
		byte messagelenlow=messagesize.byteValue();
		messagesize=messagesize.rotateRight(messagesize,8);
		byte messagelenhigh=messagesize.byteValue();
			byte[] entete=new byte[8];
		entete[0]=firstByte;
		entete[1]=SID[0];
		entete[2]=messagelenhigh;
		entete[3]=messagelenlow;
		entete[4]=sequenceNumber[0];
		entete[5]=sequenceNumber[1];
		entete[6]=timestamphigh;
		entete[7]=timestamplow;

		byte[] HE1=new byte[HEdatalength];
		HE1[0]=exthigh;
		HE1[1]=extlow1;
		HE1[2]=HEdatahigh;
		HE1[3]=HEdatalow;
		messagesize=12;
		byte[] svspmessage=new byte[messagesize];
		System.arraycopy(entete,0,svspmessage,0,entete.length);
		System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
```

```
                Long checksum=cis.getChecksum().getValue();
                byte bb1=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb2=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb3=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb4=checksum.byteValue();
                byte[] bb={bb1,bb2,bb3,bb4};

                byte[] entete2=new byte[12];
                System.arraycopy(entete,0,entete2,0,entete.length);
                System.arraycopy(bb,0,entete2,8,4);
                messagesize=16;
                svspmessage=new byte[messagesize];
                System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);

                Mac mac=Mac.getInstance("HmacSHA1");
                mac.init(secretkey);
                byte[]mactag=mac.doFinal(svspmessage);
                //System.out.println("mactag"+mactag.length);
                byte[] message=new byte[(mactag.length+messagesize)];
                System.arraycopy(svspmessage,0,message,0,messagesize);
                System.arraycopy(mactag,0,message,messagesize,mactag.length);
                return (message);
        }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_ESMSind
{
	public byte[] buildMessage(String ID,byte[] rnd,byte[] SID,byte[] sequenceNumber,SecretKey
secretkey ) throws Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=96;
		byte extlow1=0;
		int HEdatalength=SID.length;
		HEdatalength=HEdatalength+4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();

		byte[] IDtoByte=ID.getBytes();
		byte[] token=new byte[IDtoByte.length+rnd.length];
		System.arraycopy(IDtoByte,0,token,0,IDtoByte.length);
		System.arraycopy(rnd,0,token,IDtoByte.length,rnd.length);

		Integer messagesize=12+HEdatalength+token.length;
		byte messagelenlow=messagesize.byteValue();
		messagesize=messagesize.rotateRight(messagesize,8);
		byte messagelenhigh=messagesize.byteValue();
		byte[] entete=new byte[8];
		entete[0]=firstByte;
		entete[1]=SID[0];
		entete[2]=messagelenhigh;
		entete[3]=messagelenlow;
		entete[4]=sequenceNumber[0];
		entete[5]=sequenceNumber[1];
		entete[6]=timestamphigh;
		entete[7]=timestamplow;

		byte[] HE1=new byte[HEdatalength];
		HE1[0]=exthigh;
		HE1[1]=extlow1;
		HE1[2]=HEdatahigh;
		HE1[3]=HEdatalow;
```

```
                System.arraycopy(SID,0,HE1,4,SID.length);
                messagesize=8+HEdatalength+token.length;
                byte[] svspmessage=new byte[messagesize];
                System.arraycopy(entete,0,svspmessage,0,entete.length);
                System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);
                System.arraycopy(token,0,svspmessage,(entete.length+HE1.length),token.length);
                CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                byte[] buf=new byte[32];
                while(cis.read(buf)>0){}
                Long checksum=cis.getChecksum().getValue();
                byte bb1=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb2=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb3=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb4=checksum.byteValue();
                byte[] bb={bb1,bb2,bb3,bb4};

                byte[] entete2=new byte[12];
                System.arraycopy(entete,0,entete2,0,entete.length);
                System.arraycopy(bb,0,entete2,8,4);
                messagesize=12+HEdatalength+token.length;
                svspmessage=new byte[messagesize];
                System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);
                System.arraycopy(token,0,svspmessage,(entete2.length+HE1.length),token.length);

                Mac mac=Mac.getInstance("HmacSHA1");
                mac.init(secretkey);
                byte[]mactag=mac.doFinal(svspmessage);
                byte[] message=new byte[(mactag.length+messagesize)];
                System.arraycopy(svspmessage,0,message,0,messagesize);
                System.arraycopy(mactag,0,message,messagesize,mactag.length);
                return (message);
                }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class SendU_ESMSconf
{
	public byte[] buildMessage(String ID1,String ID2,String starttime,String endtime,String
key,byte[] SID,byte[] sequenceNumber,SecretKey secretkey ) throws Exception
	{
		byte firstByte = 82;
		Date start=new Date();
		Long l=start.getTime();
		Short sh =l.shortValue();
		byte timestamplow=sh.byteValue();
		l=l.rotateRight(l,8);
		sh=l.shortValue();
		byte timestamphigh=sh.byteValue();
		byte exthigh=112;
		byte extlow1=0;
		SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
		byte[] serial=new byte[1];
		sr.nextBytes(serial);

		String report="Session started at:"+starttime+" "+"Session ended at:"+endtime+" "+"User
ID:"+ID1+" "+"Remote user ID:"+ID2+" "+"Session ID:"+SID[0]+" "+"Type of call:"+"SIP"+" "+"Serial
Number:"+serial[0];
		byte[] messreport=report.getBytes();
		//System.out.println("report"+messreport.length);
		KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
		char[] password ={'p','a','s','s','w','o','r','d'};
		java.io.FileInputStream fis =
	new java.io.FileInputStream(".keystore");
		ks.load(fis, password);
	fis.close();
	char[]	keypassword ={'1','2','3','4','5','6'};
		PrivateKey myPrivateKey = (PrivateKey)ks.getKey(key, keypassword);
		Signature rsa=Signature.getInstance("MD5withRSA");
		rsa.initSign(myPrivateKey);
		rsa.update(messreport);
		byte[] sig=rsa.sign();
		//System.out.println("signature"+sig.length);
		int HEdatalength=sig.length;
		HEdatalength=HEdatalength+4;
		Integer HEdatalen=HEdatalength;
		byte HEdatalow=HEdatalen.byteValue();
		HEdatalen=HEdatalen.rotateRight(HEdatalen,8);
		byte HEdatahigh=HEdatalen.byteValue();
		Integer messagesize=12+HEdatalength+messreport.length;
		byte messagelenlow=messagesize.byteValue();
		messagesize=messagesize.rotateRight(messagesize,8);
```

```
                byte messagelenhigh=messagesize.byteValue();
                byte[] entete=new byte[8];
                entete[0]=firstByte;
                entete[1]=SID[0];
                entete[2]=messagelenhigh;
                entete[3]=messagelenlow;
                entete[4]=sequenceNumber[0];
                entete[5]=sequenceNumber[1];
                entete[6]=timestamphigh;
                entete[7]=timestamplow;

                byte[] HE1=new byte[HEdatalength];
                HE1[0]=exthigh;
                HE1[1]=extlow1;
                HE1[2]=HEdatahigh;
                HE1[3]=HEdatalow;
                System.arraycopy(sig,0,HE1,4,sig.length);

                messagesize=8+HEdatalength+messreport.length;
                byte[] svspmessage=new byte[messagesize];
                System.arraycopy(entete,0,svspmessage,0,entete.length);
                System.arraycopy(HE1,0,svspmessage,entete.length,HE1.length);

        System.arraycopy(messreport,0,svspmessage,(entete.length+HE1.length),messreport.length);

                CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                byte[] buf=new byte[32];
                while(cis.read(buf)>0){}
                Long checksum=cis.getChecksum().getValue();
                byte bb1=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb2=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb3=checksum.byteValue();
                checksum=checksum.rotateRight(checksum,8);
                byte bb4=checksum.byteValue();
                byte[] bb={bb1,bb2,bb3,bb4};

                byte[] entete2=new byte[12];
                System.arraycopy(entete,0,entete2,0,entete.length);
                System.arraycopy(bb,0,entete2,8,4);
                messagesize=12+HEdatalength+messreport.length;
                svspmessage=new byte[messagesize];
                System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                System.arraycopy(HE1,0,svspmessage,entete2.length,HE1.length);

        System.arraycopy(messreport,0,svspmessage,(entete2.length+HE1.length),messreport.length);


                Mac mac=Mac.getInstance("HmacSHA1");
                mac.init(secretkey);
                byte[]mactag=mac.doFinal(svspmessage);
        //      System.out.println("mac"+mactag.length);
                byte[] message=new byte[(mactag.length+messages ize)];
                System.arraycopy(svspmessage,0,message,0,messagesize);
```

```
                System.arraycopy(mactag,0,message,messagesize,mactag.length);
                return (message);

        }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class ReceiveU_RUCArep
{
	public byte[] getKey(String cername,String key,byte[] rnd,byte[] message) throws Exception
	{
		FileInputStream fis=new FileInputStream(cername);
		CertificateFactory cf=CertificateFactory.getInstance("X.509");
		Collection c=cf.generateCertificates(fis);
		Iterator i=c.iterator();
		java.security.cert.Certificate cert = (java.security.cert.Certificate)i.next();
		fis.close();
		PublicKey publickey=cert.getPublicKey();
		Signature md5rsa = Signature.getInstance("MD5withRSA");
		md5rsa.initVerify(publickey);
		byte[] svsp=new byte[(message.length-128)];
		byte[] signature=new byte[128];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,signature,0,128);
		md5rsa.update(svsp);
		boolean verifies = md5rsa.verify(signature);
		System.out.println("signature verifies: " + verifies);
		svsp=new byte[(message.length-132)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);
		int keysize=message.length-296;
		byte[] hres=new byte[20];
		System.arraycopy(message,16,hres,0,20);
		byte[] cipherID=new byte[128];
		System.arraycopy(message,(40+keysize),cipherID,0,128);
```

```
                    KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
                    char[] password ={'p','a','s','s','w','o','r','d'};
                    java.io.FileInputStream fis1 =
      new java.io.FileInputStream(".keystore");
                    ks.load(fis1, password);
          fis1.close();
          char[]    keypassword ={'1','2','3','4','5','6'};
                    PrivateKey myPrivateKey = (PrivateKey)ks.getKey(key, keypassword);
                    Cipher   rsaCipher = Cipher.getInstance("RSA");
                    rsaCipher.init(Cipher.DECRYPT_MODE, myPrivateKey);
                    byte[] ID=rsaCipher.doFinal(cipherID);
                    byte[] IDrnd=new byte[(ID.length+rnd.length)];
                    System.arraycopy(rnd,0,IDrnd,0,rnd.length);
                    System.arraycopy(ID,0,IDrnd,rnd.length,ID.length);
                    MessageDigest md = MessageDigest.getInstance("SHA1");
                    md.update(IDrnd);
                    byte[] hres1 = md.digest();
                    verifies=Arrays.equals(hres,hres1);
                    System.out.println("hres verifies: " + verifies);
                    byte[] dhkey=new byte[keysize];
                    System.arraycopy(message,40,dhkey,0,keysize);
                    return(dhkey);



          }
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_RUCA
{	public byte[] getRandom(String cername,byte[] message,int certsize) throws Exception
	{	FileInputStream fis=new FileInputStream(cername);
		CertificateFactory cf=CertificateFactory.getInstance("X.509");
		Collection c=cf.generateCertificates(fis);
		Iterator i=c.iterator();
		java.security.cert.Certificate cert = (java.security.cert.Certificate)i.next();
		fis.close();
		PublicKey publickey=cert.getPublicKey();
		Signature md5rsa = Signature.getInstance("MD5withRSA");
		md5rsa.initVerify(publickey);
		byte[] svsp=new byte[(message.length-128)];
		byte[] signature=new byte[128];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,signature,0,128);
		md5rsa.update(svsp);
		boolean verifies = md5rsa.verify(signature);
		System.out.println("signature verifies: " + verifies);
		svsp=new byte[(message.length-132)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);
		svsp=new byte[128];
		System.arraycopy(message,(certsize+50),svsp,0,128);
		Cipher   rsaCipher = Cipher.getInstance("RSA");
	rsaCipher.init(Cipher.DECRYPT_MODE, publickey);
	svsp = rsaCipher.doFinal(svsp);
	return(svsp);

	}
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_RUCAconf
{
	public byte[] getToken(byte[] message,SecretKey secretkey) throws Exception
	{
		Mac mac=Mac.getInstance("HmacSHA1");
		mac.init(secretkey);
		byte[] svsp=new byte[(message.length-20)];
		byte[] mactag=new byte[20];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,mactag,0,20);
		byte[]mactag1=mac.doFinal(svsp);
		boolean verifies = Arrays.equals(mactag,mactag1);
		System.out.println("Hmac verifies: " + verifies);

		svsp=new byte[(message.length-24)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);
		svsp=new byte[72];
		System.arraycopy(message,49,svsp,0,72);
		return(svsp);




	}
}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_ESMSreq
{
	public void getMessage(byte[] message,SecretKey secretkey) throws Exception
	{
		Mac mac=Mac.getInstance("HmacSHA1");
		mac.init(secretkey);
		byte[] svsp=new byte[(message.length-20)];
		byte[] mactag=new byte[20];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,mactag,0,20);
		byte[]mactag1=mac.doFinal(svsp);
		boolean verifies = Arrays.equals(mactag,mactag1);
		System.out.println("Hmac verifies: " + verifies);

		svsp=new byte[(message.length-24)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);}}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_ESMSrep
{
	public void getMessage(byte[] message,SecretKey secretkey) throws Exception
	{
		Mac mac=Mac.getInstance("HmacSHA1");
		mac.init(secretkey);
		byte[] svsp=new byte[(message.length-20)];
		byte[] mactag=new byte[20];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,mactag,0,20);
		byte[]mactag1=mac.doFinal(svsp);
		boolean verifies = Arrays.equals(mactag,mactag1);
		System.out.println("Hmac verifies: " + verifies);

		svsp=new byte[(message.length-24)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);}}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_ESMSind
{
	public byte[] getToken(byte[] message,SecretKey secretkey) throws Exception
	{
		Mac mac=Mac.getInstance("HmacSHA1");
		mac.init(secretkey);
		byte[] svsp=new byte[(message.length-20)];
		byte[] mactag=new byte[20];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,mactag,0,20);
		byte[]mactag1=mac.doFinal(svsp);
		boolean verifies = Arrays.equals(mactag,mactag1);
		System.out.println("Hmac verifies: " + verifies);

		svsp=new byte[(message.length-24)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis .getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);
		svsp=new byte[72];
		System.arraycopy(message,17,svsp,0,72);
		return(svsp);}}
```

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
public class ReceiveU_ESMSconf
{
	public byte[] getReport(String cername,byte[] message,SecretKey secretkey) throws Exception
	{
		FileInputStream fis=new FileInputStream(cername);
		CertificateFactory cf=CertificateFactory.getInstance("X.509");
		Collection c=cf.generateCertificates(fis);
		Iterator i=c.iterator();
		java.security.cert.Certificate cert = (java.security.cert.Certificate)i.next();
		fis.close();
		PublicKey publickey=cert.getPublicKey();
		Signature md5rsa = Signature.getInstance("MD5withRSA");
		md5rsa.initVerify(publickey);
		byte[] svsp=new byte[(message.length-20)];
		byte[] mactag=new byte[20];
		System.arraycopy(message,0,svsp,0,svsp.length);
		System.arraycopy(message,svsp.length,mactag,0,20);
		Mac mac=Mac.getInstance("HmacSHA1");
		mac.init(secretkey);
		byte[]mactag1=mac.doFinal(svsp);
		boolean verifies = Arrays.equals(mactag,mactag1);
		System.out.println("Hmac verifies: " + verifies);


		svsp=new byte[(message.length-24)];
		System.arraycopy(message,0,svsp,0,8);
		System.arraycopy(message,12,svsp,8,(svsp.length-8));
		byte[] check=new byte[4];
		System.arraycopy(message,8,check,0,4);
		CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
		byte[] buf=new byte[32];
		while(cis.read(buf)>0){}
		Long checksum=cis.getChecksum().getValue();
		byte bb1=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb2=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb3=checksum.byteValue();
		checksum=checksum.rotateRight(checksum,8);
		byte bb4=checksum.byteValue();
		byte[] bb={bb1,bb2,bb3,bb4};
		verifies=Arrays.equals(bb,check);
		System.out.println("cheksum verifies: " + verifies);


		int reportsize=message.length-164;
```

```
                    byte[] report=new byte[reportsize];
                    System.arraycopy(message,144,report,0,reportsize);
                    byte[] signature=new byte[128];
                    System.arraycopy(message,16,signature,0,128);
                    md5rsa.update(report);
                    verifies = md5rsa.verify(signature);
                    System.out.println("signature verifies: " + verifies);
                    return(report);



          }
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;
 class SendMessage extends Thread
 {
        SecretKey sk;
        String filename;
        String filename1;
        SendMessage(SecretKey sk,String filename,String filename1)
         {
           this.sk = sk;
           this.filename=filename;
           this.filename1=filename1;
         }
            public void run()
         {
             try
         {
             BufferedReader clavier = new BufferedReader(new InputStreamReader(System.in));
          byte[] message ;
             InetAddress adresse = InetAddress.getByName("127.0.0.1");
                                DatagramPacket envoi ;
          DatagramSocket socket = new DatagramSocket();
          Cipher desCipher;
          desCipher = Cipher.getInstance("DES");
          desCipher.init(Cipher.ENCRYPT_MODE, sk);
          String ligne;
          byte[]cipher;
          byte[] clear;
          FileOutputStream fos=new FileOutputStream(filename);
          FileOutputStream fos1=new FileOutputStream(filename1);
          while(true)
          {
             ligne=clavier.readLine();
                                        clear=ligne.getBytes();
                                        cipher=desCipher.doFinal(clear);
                                        message=cipher;

                                        fos.write(clear);
                                        fos1.write(cipher);
                                        envoi=new DatagramPacket(message,message.length,adresse, 5000);
                                        socket.send(envoi);
          }
          }
          catch (Exception e)
          {
             System.out.println(e.getMessage());
          }
         }
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;
 class ReceiveMessage extends Thread
 {
        SecretKey sk;
        String filename;
        String filename1;
         ReceiveMessage(SecretKey sk,String filename,String filename1)
         {
           this.sk = sk;
           this.filename=filename;
           this.filename1=filename1;
         }
            public void run()
         {
             try
         {
          byte[] message = new byte [1000];
           Datagra mPacket packet = new DatagramPacket(message,1000);
           DatagramSocket socket = new DatagramSocket(4001);
           Cipher desCipher;
          desCipher = Cipher.getInstance("DES");
          desCipher.init(Cipher.DECRYPT_MODE, sk);
          String ligne;
          byte[]cipher;
          byte[] clear;
          FileOutputStream fos=new FileOutputStream(filename);
          FileOutputStream fos1=new FileOutputStream(filename1);
          while(true)
          {
            socket.receive(packet);
            int numBytesReceived =packet.getLength();
                                        cipher=new byte[numBytesReceived];
                                        System.arraycopy(message,0,cipher,0,numBytesReceived);
                                        System.out.println(new String(cipher));
                                        clear=desCipher.doFinal(cipher);
                                        System.out.println(new String(clear));
                                        fos.write(clear);
                                        fos1.write(cipher);
          }
          }
          catch (Exception e)
          {
            System.out.println(e.getMessage());
          }

         }
}
```

```
import java.awt.*;
import java.awt.event.*;
public class Endcallpanel extends Frame implements ActionListener
{
	Button b;
	public Endcallpanel(){
	setLayout(new GridLayout(2,3));
	b = new Button("EndCall");
	b.addActionListener(this);
	add(b);

	}
	public void actionPerformed(ActionEvent e)
	{
	hide();
	}
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;

public class UDPUAclient
{
        static final int port = 10000;

        static public void main (String[] args) throws Exception
        {

                InetAddress adresse = InetAddress.getByName("127.0.0.1");
                DatagramPacket udppacket;
                DatagramSocket socket = new DatagramSocket();
                System.out.println("Sending U_RUCAreq ...");
                Date start=new Date();
                String start_time=start.toString();
                SendU_RUCAreq request=new SendU_RUCAreq();
                SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
                byte[] randomA=new byte[64];
                sr.nextBytes(randomA);
                byte[] SID =new byte[1];
                sr.nextBytes(SID);
                byte[] sequenceNumber=new byte[2];
                sr.nextBytes(sequenceNumber);
                byte[]
mess=request.buildMessage("tonycert.cer","tonykey",randomA,SID,sequenceNumber);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_RUCAreq ...");
                mess=new byte[902];
                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);

                ReceiveU_RUCA receive1=new ReceiveU_RUCA();
                byte[] receivedmess=receive1.getRandom("carolecert.cer",mess,596);
                byte[] randomB=receivedmess;

                System.out.println("Sending U_RUCArep ...");
                randomA[0]=(byte)128;
                BigInteger p = new BigInteger(1,randomA);
        BigInteger g = new BigInteger(1,randomB);
        int exp =511;

        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
    DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
    keyGen.initialize(dhSpec);
    KeyPair keypair = keyGen.generateKeyPair();
    // Get the generated public and private keys
    PublicKey DHpublicKey = keypair.getPublic();
```

```
PrivateKey DHprivateKey = keypair.getPrivate();

sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCArep request2=new SendU_RUCArep();

    mess=request2.buildMessage("12345678","tonykey","carolecert.cer",SID,sequenceNumber,rando
mB,DHpublicKey);

            udppacket = new DatagramPacket(mess,mess.length,adresse, port);
            socket.send(udppacket);
            System.out.println("Receiving U_RUCArep ...");

            mess=new byte[1000];

            udppacket = new DatagramPacket(mess,mess.length);
            socket.receive(udppacket);
            byte[] mess1=new byte[udppacket.getLength()];
System.arraycopy(mess,0,mess1,0,mess1.length);
            ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
            receivedmess=receive2.getKey("carolecert.cer","tonykey",randomA,mess1);
            X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
KeyFactory keyFact = KeyFactory.getInstance("DH");
DHpublicKey = keyFact.generatePublic(x509KeySpec);
KeyAgreement ka = KeyAgreement.getInstance("DH");
ka.init(DHprivateKey);
ka.doPhase(DHpublicKey, true);
  String algorithm = "DES";

// Generate the secret key
SecretKey secretKey1 = ka.generateSecret(algorithm);
System.out.println("Sending U_RUCAconf ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCAconf request3=new SendU_RUCAconf();
            mess=request3.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
            udppacket = new DatagramPacket(mess,mess.length,adresse, port);
            socket.send(udppacket);

            System.out.println("Receiving U_RUCAconf...");
            mess=new byte[141];
            udppacket = new DatagramPacket(mess,mess.length);
            socket.receive(udppacket);

            ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
            receivedmess=receive3.getToken(mess,secretKey1);
            byte[] TokenB=receivedmess;
            System.out.println("Setup completed!");
            SendMessage sm=new
SendMessage(secretKey1,"client\\clearoutputlog.txt","client\\cipheroutputlog.txt");
 ReceiveMessage rm=new
ReceiveMessage(secretKey1,"client\\clearinputlog.txt","client\\cipherinputlog.txt");
    rm.start();
sm.start();
```

```
    Endcallpanel f= new Endcallpanel();
    f.show();
while(f.isShowing());
System.out.println("Call ended!");
sm.stop();
rm.stop();
byte[] endcallmess="endcall".getBytes();
adresse=InetAddress.getByName("127.0.0.1");
DatagramPacket envoi=new DatagramPacket(endcallmess,endcallmess.length,adresse, 6000);
DatagramSocket socket1 = new DatagramSocket();
            socket1.send(envoi);
            socket1.close();
            System.out.println("Sending U_ESMSreq...");
            sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSreq request4=new SendU_ESMSreq();
            mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
            udppacket = new DatagramPacket(mess,mess.length,adresse, port);
            socket.send(udppacket);

            System.out.println("Receiving U_ESMSreq...");

            mess=new byte[36];

            udppacket = new DatagramPacket(mess,mess.length);
            socket.receive(udppacket);
            ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
            receive4.getMessage(mess,secretKey1);
            System.out.println("Sending U_ESMSrep...");
            sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSrep request5=new SendU_ESMSrep();
            mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
            udppacket = new DatagramPacket(mess,mess.length,adresse, port);
            socket.send(udppacket);

            System.out.println("Receiving U_ESMSrep...");
            mess=new byte[36];

            udppacket = new DatagramPacket(mess,mess.length);
            socket.receive(udppacket);
            ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
            receive5.getMessage(mess,secretKey1);
            System.out.println("Sending U_ESMSind...");
            sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSind request6=new SendU_ESMSind();
            mess=request6.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
            udppacket = new DatagramPacket(mess,mess.length,adresse, port);
            socket.send(udppacket);

            System.out.println("Receiving U_ESMSind...");
```

```
                    mess=new byte[109];

                    udppacket = new DatagramPacket(mess,mess.length);
                    socket.receive(udppacket);
                    ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
                    receivedmess=receive6.getToken(mess,secretKey1);
                    String rec=new String(receivedmess);
                    System.out.println("Returned Tokon:"+rec);
                    System.out.println("Sending U_ESMSconf...");
                    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSconf request7=new SendU_ESMSconf();
                    Date end=new Date();
                    String end_time=end.toString();

        mess=request7.buildMessage("12345678","87654321",start_time,end_time,"tonykey",SID,sequen
ceNumber,secretKey1);
                    udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                    socket.send(udppacket);

                    System.out.println("Receiving U_ESMSconf...");

                    mess=new byte[1000];
                    udppacket = new DatagramPacket(mess,mess.length);
                    socket.receive(udppacket);
                    mess1=new byte[udppacket.getLength()];
    System.arraycopy(mess,0,mess1,0,mess1.length);
                    ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
                    receivedmess=receive7.getReport("carolecert.cer",mess1,secretKey1);
                    FileOutputStream fos=new FileOutputStream("client\\clientreportlog.txt");
                    fos.write(receivedmess);
                    fos.close();
                    rec=new String(receivedmess);
                    System.out.println("Distante user report:"+rec);
    socket.close();
    System.exit(0);

        }
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;

public class UDPUAserver
{
        static final int port = 10000;
        static public void main (String[] args) throws Exception
        {

                DatagramSocket socket = new DatagramSocket(port);
                DatagramPacket udppacket;
                byte[] mess=new byte[926];
                udppacket = new DatagramPacket (mess,mess.length);
                System.out.println("Waiting for connection...");
                socket.receive(udppacket);
                int po=udppacket.getPort();
                InetAddress ad=udppacket.getAddress();
                System.out.println("Client connected!");
                System.out.println(" waiting for U_RUCAreq ... ");
                ReceiveU_RUCA receive1=new ReceiveU_RUCA();
                byte[] receivedmess=receive1.getRandom("tonycert.cer",mess,620);

                System.out.println("Sending U_RUCAreq...");
                byte[] randomA=receivedmess;
                Date start=new Date();
                String start_time=start.toString();
                SendU_RUCAreq request=new SendU_RUCAreq();
                SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
                byte[] randomB=new byte[64];
                sr.nextBytes(randomB);
                byte[] SID =new byte[1];
                sr.nextBytes(SID);
                byte[] sequenceNumber=new byte[2];
                sr.nextBytes(sequenceNumber);

        mess=request.buildMessage("carolecert.cer","carolekey",randomB,SID,sequenceNumber);

                randomA[0]=(byte)128;
                BigInteger p = new BigInteger(1,randomA);
        BigInteger g = new BigInteger(1,randomB);
        int exp =511;

        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
    DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
    keyGen.initialize(dhSpec);
    KeyPair keypair = keyGen.generateKeyPair();

    PublicKey DHpublicKey = keypair.getPublic();
    PrivateKey DHprivateKey = keypair.getPrivate();
    udppacket = new DatagramPacket (mess,mess.length,ad,po);
```

```
socket.send(udppacket);

System.out.println("Receiving U_RUCA.rep ...." );

                mess=new byte[1000];
                udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);
byte[] mess1=new byte[udppacket.getLength()];
System.arraycopy(mess,0,mess1,0,mess1.length);

                ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
                receivedmess=receive2.getKey("tonycert.cer","carolekey",randomB,mess1);
                X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
KeyFactory keyFact = KeyFactory.getInstance("DH");
PublicKey publicKey = keyFact.generatePublic(x509KeySpec);

KeyAgreement ka = KeyAgreement.getInstance("DH");
ka.init(DHprivateKey);
ka.doPhase(publicKey, true);
String algorithm = "DES";

// Generate the secret key
SecretKey secretKey1 = ka.generateSecret(algorithm);
System.out.println("Sending U_RUCArep ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCArep request2=new SendU_RUCArep();

    mess=request2.buildMessage("87654321","carolekey","tonycert.cer",SID,sequenceNumber,rando
mA,DHpublic Key);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);

                System.out.println("Receiving U_RUCAconf ...");
                mess=new byte[141];
                udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);

                ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
                receivedmess=receive3.getToken(mess,secretKey1);
                byte[] TokenB=receivedmess;
                System.out.println("Sending U_RUCAconf ...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCAconf request3=new SendU_RUCAconf();
                mess=request3.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);

                System.out.println("Setup completed!");
                SendMessage1 sm=new
SendMessage1(secretKey1,"server\\clearoutputlog.txt","server\\cipheroutputlog.txt");
ReceiveMessage1 rm=new
ReceiveMessage1(secretKey1,"server\\clearinputlog.txt","server\\cipherinputlog.txt");
```

```
        rm.start();
    sm.start();
                boolean endcall=false;
                byte[] endcallmess=new byte["endcall".getBytes().length];
                DatagramPacket packet=new DatagramPacket(endcallmess,endcallmess.length);
                DatagramSocket udpsocket = new DatagramSocket(6000);
                do
                {
                        udpsocket.receive(packet);
                        if(new String(endcallmess)!="")
                        endcall=true;
                }
                while(endcall==false);
                udpsocket.close();
                sm.stop();
                rm.stop();
                System.out.println("Distante user ended the call!");
                System.out.println("Receiving U_ESMSreq ...");
                mess=new byte[36];
                udppacket = new DatagramPacket (mess,mess.length);
    socket.receive(udppacket);

                ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
                receive4.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSreq ...");
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSreq request4=new SendU_ESMSreq();
                mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
    socket.send(udppacket);

                System.out.println("Receiving U_ESMSrep ...");
                mess=new byte[36];
                udppacket = new DatagramPacket (mess,mess.length);
    socket.receive(udppacket);

                ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
                receive5.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSrep ...");
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSrep request5=new SendU_ESMSrep();
                mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
    socket.send(udppacket);

                System.out.println("Receiving U_ESMSind ...");
                mess=new byte[109];
                udppacket = new DatagramPacket (mess,mess.length);
    socket.receive(udppacket);

                ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
                receivedmess=receive6.getToken(mess,secretKey1);
```

```
                    String rec=new String(receivedmess);
                    System.out.println("Returned Tokon:"+rec);
                    System.out.println("Sending U_ESMSind ...");
                    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSind request6=new SendU_ESMSind();
                    mess=request6.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
                    udppacket = new DatagramPacket (mess,mess.length,ad,po);
    socket.send(udppacket);

                    System.out.println("Receiving U_ESMSconf ...");

                    mess=new byte[1000];
                    udppacket = new DatagramPacket (mess,mess.length);
    socket.receive(udppacket);
                    mess1=new byte[udppacket.getLength()];
    System.arraycopy(mess,0,mess1,0,mess1.length);
                    ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
                    receivedmess=receive7.getReport("tonycert.cer",mess1,secretKey1);
                    FileOutputStream fos=new FileOutputStream("server\\serverreportlog.txt");
                    fos.write(receivedmess);
                    fos.close();
                    rec=new String(receivedmess);
                    System.out.println("Distante user report:"+rec);
                    System.out.println("Sending U_ESMSconf ...");
                    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSconf request7=new SendU_ESMSconf();
                    Date end=new Date();
                    String end_time=end.toString();

          mess=request7.buildMessage("87654321","12345678",start_time,end_time,"carolekey",SID,seque
nceNumber,secretKey1);
                    udppacket = new DatagramPacket (mess,mess.length,ad,po);
    socket.send(udppacket);

                    socket.close();
                    System.exit(0);


          }
}
```

```
import com.ibm.media.codec.audio.AudioCodec;
import com.sun.media.BasicPlugIn;
import javax.media.*;
import javax.media.format.AudioFormat;
import javax.crypto.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;


public class Customg723packetizer extends CustomAudioPacketizer
{

   public Customg723packetizer(SecretKey secretkey,byte[] SID,byte[] sequenceNumber)
   {        super.secretkey=secretkey;
            super.SID=SID;
            super.sequenceNumber=sequenceNumber;
      super.packetSize = 48;
      super.supportedInputFormats = (new AudioFormat[] {
         new AudioFormat("g723", 8000D, -1, 1, -1, -1, 192, -1D, Format.byteArray)
      });
      super.defaultOutputFormats = (new AudioFormat[] {
         new AudioFormat("g723/rtp", 8000D, -1, 1, -1, -1, 192, -1D, Format.byteArray)
      });
      super.PLUGIN_NAME = "G723 Packetizer";
   }

   protected Format[] getMatchingOutputFormats(Format in)
   {
      AudioFormat af = (AudioFormat)in;
      super.supportedOutputFormats = (new AudioFormat[] {
         new AudioFormat("g723/rtp", 8000D, -1, 1, -1, -1, 192, -1D, Format.byteArray)
      });
      return super.supportedOutputFormats;
   }

   public void open()
      throws ResourceUnavailableException
   {
      setPacketSize(super.packetSize);
      reset();
   }

   public Object[] getControls()
   {
      if(super.controls == null)
      {
         super.controls = new Control[1];
         super.controls[0] = new PacketSizeAdapter(this, super.packetSize, true);
      }
      return (Object[])super.controls;
   }

   public synchronized void setPacketSize(int newPacketSize)
   {
      super.packetSize = newPacketSize;
```

```
        super.sample_count = (super.packetSize / 24) * 240;
        if(super.history == null)
        {
            super.history = new byte[super.packetSize];
            return;
        }
        if(super.packetSize > super.history.length)
        {
            byte newHistory[] = new byte[super.packetSize];
            System.arraycopy(super.history, 0, newHistory, 0, super.historyLength);
            super.history = newHistory;
        }
    }
}
```

```
import com.sun.media.BasicCodec;
import com.sun.media.BasicPlugIn;
import javax.media.Buffer;
import javax.crypto.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;
import java.io.*;


public abstract class CustomAudioPacketizer extends AudioCodec
{
        SecretKey secretkey;
        byte[] SID;
        byte[] sequenceNumber;
  public CustomAudioPacketizer()
  {

  }

  public synchronized int process(Buffer inputBuffer, Buffer outputBuffer)
  {


                try{

                Date start=new Date();
                Long l=start.getTime();
                byte firstByte = 82;
                Short sh =l.shortValue();
                byte timestamplow=sh.byteValue();
                l=l.rotateRight(l,8);
                sh=l.shortValue();
                byte timestamphhigh=sh.byteValue();
                l=l.rotateLeft(l,8);
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    Integer messagesize=60;
    byte messagelenlow=messagesize.byteValue();
                messagesize=messagesize.rotateRight(messagesize,8);
                byte messagelenhigh=messagesize.byteValue();
                messagesize=60;
                byte[] entete=new byte[8];
                entete[0]=firstByte;
                entete[1]=SID[0];
                entete[2]=messagelenhigh;
                entete[3]=messagelenlow;
                entete[4]=sequenceNumber[0];
                entete[5]=sequenceNumber[1];
```

```
                    entete[6]=timestamphigh;
                    entete[7]=timestamplow;
                    Cipher desCipher;
            desCipher = Cipher.getInstance("DES/ECB/NoPadding");
            desCipher.init(Cipher.ENCRYPT_MODE, secretkey);

      int inpLength = inputBuffer.getLength();
      int outLength = packetSize;
      byte inpData[] = (byte[])inputBuffer.getData();
      byte outData[] = validateByteArraySize(outputBuffer, outLength+32);
      if(inpLength + historyLength >= packetSize)
      {
         int copyFromHistory = Math.min(historyLength, packetSize);
         System.arraycopy(history, 0, outData, 0, copyFromHistory);
         int remainingBytes = packetSize - copyFromHistory;
         System.arraycopy(inpData, inputBuffer.getOffset(), outData, historyLength, remainingBytes);
         historyLength -= copyFromHistory;
         inputBuffer.setOffset(inputBuffer.getOffset() + remainingBytes);
         inputBuffer.setLength(inpLength - remainingBytes);
         try{
         outData=desCipher.doFinal(outData);

         }
         catch (BadPaddingException e){System.out.println("bad padding!");}
         catch (IllegalBlockSizeException e){System.out.println("blok size!");}
           byte[] svspmessage=new byte[56];
           System.arraycopy(entete,0,svspmessage,0,entete.length);
                              System.arraycopy(outData,0,svspmessage,entete.length,48);
                              CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                              byte[] buf=new byte[32];
                              try{
                              while(cis.read(buf)>0){}}
                              catch(IOException e){}
                              Long checksum=cis.getChecksum().getValue();
                              byte bb1=checksum.byteValue();
                              checksum=checksum.rotateRight(checksum,8);
                              byte bb2=checksum.byteValue();
                              checksum=checksum.rotateRight(checksum,8);
                              byte bb3=checksum.byteValue();
                              checksum=checksum.rotateRight(checksum,8);
                              byte bb4=checksum.byteValue();
                              byte[] bb={bb1,bb2,bb3,bb4};

                              byte[] entete2=new byte[12];
                              System.arraycopy(entete,0,entete2,0,entete.length);
                              System.arraycopy(bb,0,entete2,8,4);
                              svspmessage=new byte[messagesize];
                              System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                              System.arraycopy(outData,0,svspmessage,entete2.length,48);
                              Mac mac=Mac.getInstance("HmacSHA1");
                              mac.init(secretkey);
                              byte[]mactag=mac.doFinal(svspmessage);
                              byte[] message=new byte[(mactag.length+messagesize)];
                              System.arraycopy(svspmessage,0,message,0,messagesize);
                              System.arraycopy(mactag,0,message,messagesize,mactag.length);
```

```
        outputBuffer.setData(message);
        updateOutput(outputBuffer, super.outputFormat, message.length, 0);
        return 2;
      }
      if(inputBuffer.isEOM())
      {
        System.arraycopy(history, 0, outData, 0, historyLength);
        System.arraycopy(inpData, inputBuffer.getOffset(), outData, historyLength, inpLength);
        try {outData=desCipher.doFinal(outData);}
          catch (BadPaddingException e){}
        catch (IllegalBlockSizeException e){System.out.println("blok size!");}
          outData=inpData;
          byte[] svspmessage=new byte[messagesize-4];
                                System.arraycopy(entete,0,svspmessage,0,entete.length);
                                System.arraycopy(outData,0,svspmessage,entete.length,48);
                                CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svspmessage),new CRC32());
                                byte[] buf=new byte[32];
                                try{
                                while(cis.read(buf)>0){}}
                                catch(IOException e){}
                                Long checksum=cis.getChecksum().getValue();
                                byte bb1=checksum.byteValue();
                                checksum=checksum.rotateRight(checksum,8);
                                byte bb2=checksum.byteValue();
                                checksum=checksum.rotateRight(checksum,8);
                                byte bb3=checksum.byteValue();
                                checksum=checksum.rotateRight(checksum,8);
                                byte bb4=checksum.byteValue();
                                byte[] bb={bb1,bb2,bb3,bb4};

                                byte[] entete2=new byte[12];
                                System.arraycopy(entete,0,entete2,0,entete.length);
                                System.arraycopy(bb,0,entete2,8,4);
                                svspmessage=new byte[messagesize];
                                System.arraycopy(entete2,0,svspmessage,0,entete2.length);
                                System.arraycopy(outData,0,svspmessage,entete2.length,48);
                                Mac mac=Mac.getInstance("HmacSHA1");
                                mac.init(secretkey);
                                byte[]mactag=mac.doFinal(svspmessage);
                                byte[] message=new byte[(mactag.length+messagesize)];
                                System.arraycopy(svspmessage,0,message,0,messagesize);
                                System.arraycopy(mactag,0,message,messagesize,mactag.length);
        outputBuffer.setData(message);
        updateOutput(outputBuffer, super.outputFormat, message.length, 0);
        historyLength = 0;
        return 0;
      } else
      {
        System.arraycopy(inpData, inputBuffer.getOffset(), history, historyLength, inpLength);
        historyLength += inpLength;
        return 4;
      }}
      catch (NoSuchPaddingException e){System.out.println("no padding!");}
            catch (NoSuchAlgorithmException e){System.out.println("no alghorithm!");}
            catch (InvalidKeyException e){System.out.println("wrong key!");}
```

```
            return 4;
    }

    public void reset()
    {
        historyLength = 0;
    }

    protected byte history[];
    protected int packetSize;
    protected int historyLength;
    protected int sample_count;
}
```

```
import com.sun.media.BasicCodec;
import com.sun.media.BasicPlugIn;
import com.sun.media.controls.SilenceSuppressionAdapter;
import javax.media.*;
import javax.media.format.AudioFormat;
import javax.crypto.*;
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import java.math.*;
import java.lang.*;
import java.lang.reflect.*;

public class Customg723JavaDecoder extends AudioCodec
{
            SecretKey secretkey;
   public Customg723JavaDecoder(SecretKey deskey)
   {        this.secretkey=deskey;
      super.supportedInputFormats = (new AudioFormat[] {
         new AudioFormat("g723"), new AudioFormat("g723/rtp")
      });
      super.defaultOutputFormats = (new AudioFormat[] {
         new AudioFormat("LINEAR")
      });
      super.PLUGIN_NAME = "G723 Decoder";
   }

   protected Format[] getMatchingOutputFormats(Format in)
   {
      AudioFormat af = (AudioFormat)in;
      super.supportedOutputFormats = (new AudioFormat[] {
         new AudioFormat("LINEAR", af.getSampleRate(), 16, 1, 0, 1)
      });
      return super.supportedOutputFormats;
   }

   public void open()
      throws ResourceUnavailableException
   {
      decoder = new G723Dec();
      decoder.decoderOpen();
   }

   public void reset()
   {
      resetDecoder();
   }

   public void close()
   {
```

```
    freeDecoder();
  }
  public int process(Buffer inputBuffer, Buffer outputBuffer)
  {
    if(!checkInputBuffer(inputBuffer))
      return 1;
    if(isEOM(inputBuffer))
    {
      propagateEOM(outputBuffer);
      return 0;
    } else
    {
                try{
                  Cipher desCipher;
                  desCipher = Cipher.getInstance("DES/ECB/NoPadding");
                  desCipher.init(Cipher.DECRYPT_MODE, secretkey);
                  Mac mac=Mac.getInstance("HmacSHA1");
                        mac.init(secretkey);
      int inpLength = inputBuffer.getLength();
      byte inpData[] = (byte[])inputBuffer.getData();
      byte[] svsp=new byte[60];
      System.arraycopy(inpData,12,svsp,0,60);
      byte[] mactag=new byte[20];
      System.arraycopy(inpData,72,mactag,0,20);
      byte[]mactag1=mac.doFinal(svsp);
                        boolean verifies = Arrays.equals(mactag,mactag1);
                        System.out.println("Hmac verifies: " + verifies);
      svsp=new byte[56];
      System.arraycopy(inpData,12,svsp,0,8);
      System.arraycopy(inpData,24,svsp,8,48);
      byte[] check=new byte[4];
                        System.arraycopy(inpData,20,check,0,4);
                        CheckedInputStream cis=new CheckedInputStream(new
ByteArrayInputStream(svsp),new CRC32());
                        byte[] buf=new byte[32];
                        try{
                        while(cis.read(buf)>0){}}
                        catch(IOException e){}
                        Long checksum=cis.getChecksum().getValue();
                        byte bb1=checksum.byteValue();
                        checksum=checksum.rotateRight(checksum,8);
                        byte bb2=checksum.byteValue();
                        checksum=checksum.rotateRight(checksum,8);
                        byte bb3=checksum.byteValue();
                        checksum=checksum.rotateRight(checksum,8);
                        byte bb4=checksum.byteValue();
                        byte[] bb={bb1,bb2,bb3,bb4};
                        verifies=Arrays.equals(bb,check);
                        System.out.println("cheksum verifies: " + verifies);

      svsp=new byte[48];
      System.arraycopy(inpData,24,svsp,0,48);
      svsp=desCipher.doFinal(svsp);


      inpData=new byte[2000];
```

```
            System.arraycopy(svsp,0,inpData,12,48);
            byte outData[] = validateByteArraySize(outputBuffer,960);
            decode(inpData, inputBuffer.getOffset(), outData, 0,48);
            updateOutput(outputBuffer, super.outputFormat,960, 0);}
            catch (NoSuchPaddingException e){System.out.println("no padding!");}
                                catch (NoSuchAlgorithmException e){System.out.println("no alghorithm!");}
                                catch (InvalidKeyException e){System.out.println("wrong key!");}
                                catch (BadPaddingException e){System.out.println("bad padding!");}
            catch (IllegalBlockSizeException e){System.out.println("blok size!");}
            return 0;
        }
    }

    protected void initDecoder()
    {
        decoder.decoderReset();
    }

    protected void freeDecoder()
    {
        decoder = null;
    }
    protected void resetDecoder()
    {
        decoder.decoderReset();
    }
    protected int calculateOutputSize(int inputSize)
    {
        return (inputSize / 24) * 480;
    }
    protected void decode(byte inpData[], int readPtr, byte outData[], int writePtr, int inpLength)
    {
        int numberOfFrames = inpLength / 24;
        int frameSize = 24;
        for(int n = 0; n < numberOfFrames;)
        {
            decoder.decodeFrame(inpData, readPtr, outData, writePtr);
            n++;
            readPtr += frameSize;
            writePtr += 480;
        }
    }
    public Object[] getControls()
    {
        if(super.controls == null)
        {
            super.controls = new Control[1];
            super.controls[0] = new SilenceSuppressionAdapter(this, true, false);
        }
        return (Object[])super.controls;
    }
    protected G723Dec decoder;
}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.control.*;
import javax.media.format.*;
import com.ibm.media.codec.audio.ulaw.*;
import com.sun.media.codec.audio.ulaw.*;
import com.ibm.media.codec.audio.rc.*;
import com.sun.media.ui.*;

public class Voiceudpuaclient
{
	static final int port = 10000;

	static public void main (String[] args) throws Exception
	{

		InetAddress adresse = InetAddress.getByName("127.0.0.1");
		DatagramPacket udppacket;
		DatagramSocket socket = new DatagramSocket();
		System.out.println("Sending U_RUCAreq ...");
		Date start=new Date();
		String start_time=start.toString();
		long timebefore=start.getTime();
		SendU_RUCAreq request=new SendU_RUCAreq();
		SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
		byte[] randomA=new byte[64];
		sr.nextBytes(randomA);
		byte[] SID =new byte[1];
		sr.nextBytes(SID);
		byte[] sequenceNumber=new byte[2];
		sr.nextBytes(sequenceNumber);
		byte[]
mess=request.buildMessage("tonycert.cer","tonykey",randomA,SID,sequenceNumber);
		udppacket = new DatagramPacket(mess,mess.length,adresse, port);
		socket.send(udppacket);

		System.out.println("Receiving U_RUCAreq ...");
		mess=new byte[902];
		udppacket = new DatagramPacket(mess,mess.length);
		socket.receive(udppacket);

		ReceiveU_RUCA receive1=new ReceiveU_RUCA();
		byte[] receivedmess=receive1.getRandom("carolecert.cer",mess,596);
		byte[] randomB=receivedmess;

		System.out.println("Sending U_RUCArep ...");
		randomA[0]=(byte)128;
		BigInteger p = new BigInteger(1,randomA);
```

```
        BigInteger g = new BigInteger(1,randomB);
        int exp =511;

        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
    DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
    keyGen.initialize(dhSpec);
    KeyPair keypair = keyGen.generateKeyPair();
    // Get the generated public and private keys
    PublicKey DHpublicKey = keypair.getPublic();
    PrivateKey DHprivateKey = keypair.getPrivate();

    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_RUCArep request2=new SendU_RUCArep();

        mess=request2.buildMessage("12345678","tonykey","carolecert.cer",SID,sequenceNumber,rando
mB,DHpublicKey);

                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);
                System.out.println("Receiving U_RUCArep ...");

                mess=new byte[1000];

                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);
                byte[] mess1=new byte[udppacket.getLength()];
    System.arraycopy(mess,0,mess1,0,mess1.length);
                ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
                receivedmess=receive2.getKey("carolecert.cer","tonykey",randomA,mess1);
                X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
    KeyFactory keyFact = KeyFactory.getInstance("DH");
    DHpublicKey = keyFact.generatePublic(x509KeySpec);
    KeyAgreement ka = KeyAgreement.getInstance("DH");
    ka.init(DHprivateKey);
    ka.doPhase(DHpublicKey, true);
      String algorithm = "DES";

    // Generate the secret key
    SecretKey secretKey1 = ka.generateSecret(algorithm);
    System.out.println("Sending U_RUCAconf ...");
    sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_RUCAconf request3=new SendU_RUCAconf();
                mess=request3.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_RUCAconf...");
                mess=new byte[141];
                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);

                ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
```

```
                    receivedmess=receive3.getToken(mess,secretKey1);
                    byte[] TokenB=receivedmess;
                    start=new Date();
                    long timeafter=start.getTime();
                    System.out.println("Setup completed!");
                    System.out.println("Time needed to complete setup in ms:"+(timeafter-timebefore));
                    AudioFormat format= new
AudioFormat(AudioFormat.LINEAR,8000,16,1,AudioFormat.LITTLE_ENDIAN,
AudioFormat.SIGNED);
          Vector devices= CaptureDeviceManager.getDeviceList(format);
          Processor processor;
          CaptureDeviceInfo di= null;
          if (devices.size() > 0) {
          di = (CaptureDeviceInfo) devices.elementAt( 0);

          }
          else {
                              // exit if we could not find the relevant capturedevice.
                              System.exit(-1);
          }
          // Create a processor for this capturedevice & exit if we
          // cannot create it

          processor = Manager.createProcessor(di.getLocator());

          // configure the processor
          processor.configure();
          // block until it has been configured
          while(processor.getState()!=processor.Configured);
          processor.setContentDescriptor(new ContentDescriptor(ContentDescriptor.RAW));
          TrackControl tr[] = processor.getTrackControls();
          Customg723packetizer codec=new Customg723packetizer(secretKey1,SID,sequenceNumber);
          Codec[] codecs=new Codec[1];
          codecs[0]=codec;
          tr[0].setCodecChain(codecs);
          TrackControl track[] = processor.getTrackControls();


          boolean encodingOk = false;
          // Go through the tracks and try to program one of them to
          // output gsm data.
          for (int i = 0; i < track.length; i++) {
          if (!encodingOk && track[i] instanceof FormatControl) {
          if (((FormatControl)track[i]).setFormat( new AudioFormat(AudioFormat.G723_RTP,8000,8,1))
== null) {
          track[i].setEnabled(false);
          }
          else {
                    encodingOk = true;
                    System.out.println("encoding found");
                    }
                    } else {
                    // we could not set this track to gsm, so disable it
                    track[i].setEnabled(false);
                              }
          }
```

```
        // At this point, we have determined where we can send out
        // gsm data or not.
        // realize the processor
        if (encodingOk) {
                processor.realize();
                while(processor.getState()!=processor.Realized);

        DataSource ds= null;
        try {
        ds = processor.getDataOutput();
                } catch (NotRealizedError e) {
                System.exit(-1);
        }

        // hand this datasource to manager for creating an RTP
        // datasink our RTP datasimnk will multicast the audio
        try{
                String url= "rtp://127.0.0.1:59150/audio/1";
                MediaLocator m = new MediaLocator(url);
                DataSink d = Manager.createDataSink(ds, m);
                d.open();
                d.start();

                } catch (Exception e) {
                        System.out.println("error");
                System.exit(-1);
                }
        new PlayerWindow(processor);
                processor.start();

        Endcallpanel f= new Endcallpanel();
        f.show();
    while(f.isShowing());
    System.out.println("Call ended!");
    processor.close();
    byte[] endcallmess="endcall".getBytes();
    adresse=InetAddress.getByName("127.0.0.1");
    DatagramPacket envoi=new DatagramPacket(endcallmess,endcallmess.length,adresse, 6000);
    DatagramSocket socket1 = new DatagramSocket();
                socket1.send(envoi);
                socket1.close();
                System.out.println("Sending U_ESMSreq...");
                start=new Date();
                timebefore=start.getTime();
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSreq request4=new SendU_ESMSreq();
                mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_ESMSreq...");

                mess=new byte[36];
```

```
                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);
                ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
                receive4.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSrep...");
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSrep request5=new SendU_ESMSrep();
                mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_ESMSrep...");
                mess=new byte[36];

                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);
                ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
                receive5.getMessage(mess,secretKey1);
                System.out.println("Sending U_ESMSind...");
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSind request6=new SendU_ESMSind();
                mess=request6.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_ESMSind...");
                mess=new byte[109];

                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);
                ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
                receivedmess=receive6.getToken(mess,secretKey1);
                String rec=new String(receivedmess);
                System.out.println("Returned Tokon:"+rec);
                System.out.println("Sending U_ESMSconf...");
                sequenceNumber[0]++;
    if(sequenceNumber[0]==(byte)0)
    sequenceNumber[1]++;
    SendU_ESMSconf request7=new SendU_ESMSconf();
                Date end=new Date();
                String end_time=end.toString();

        mess=request7.buildMessage("12345678","87654321",start_time,end_time,"tonykey",SID,sequen
ceNumber,secretKey1);
                udppacket = new DatagramPacket(mess,mess.length,adresse, port);
                socket.send(udppacket);

                System.out.println("Receiving U_ESMSconf...");

                mess=new byte[1000];
                udppacket = new DatagramPacket(mess,mess.length);
                socket.receive(udppacket);
```

```
                    mess1=new byte[udppacket.getLength()];
    System.arraycopy(mess,0,mess1,0,mess1.length);
                    ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
                    receivedmess=receive7.getReport("carolecert.cer",mess1,secretKey1);
                    FileOutputStream fos=new FileOutputStream("client\\clientreportlog.txt");
                    fos.write(receivedmess);
                    fos.close();
                    rec=new String(receivedmess);
                    System.out.println("Distante user report:"+rec);
    socket.close();
    start=new Date();
    timeafter=start.getTime();
    System.out.println("time needed to finish session in ms:"+(timeafter-timebefore));
    System.exit(0);

        }
}}
```

```
import java.security.*;
import java.security.spec.*;
import javax.crypto.spec.*;
import javax.crypto.*;
import java.io.*;
import java.math.*;
import java.util.*;
import java.net.*;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.control.*;
import javax.media.format.*;
import com.ibm.media.codec.audio.g723.*;
import com.ibm.media.codec.audio.rc.*;
import com.sun.media.ui.*;

public class Voiceudpuaserver
{
        static final int port = 10000;
        static public void main (String[] args) throws Exception
        {
                DatagramSocket socket = new DatagramSocket(port);
                DatagramPacket udppacket;
                byte[] mess=new byte[926];
                udppacket = new DatagramPacket (mess,mess.length);
                System.out.println("Waiting for connection...");
                socket.receive(udppacket);
                int po=udppacket.getPort();
                InetAddress ad=udppacket.getAddress();
                System.out.println("Client connected!");
                System.out.println(" waiting for U_RUCAreq ... ");
                ReceiveU_RUCA receive1=new ReceiveU_RUCA();
                byte[] receivedmess=receive1.getRandom("tonycert.cer",mess,620);

                System.out.println("Sending U_RUCAreq...");
                byte[] randomA=receivedmess;
                Date start=new Date();
                String start_time=start.toString();
                SendU_RUCAreq request=new SendU_RUCAreq();
                SecureRandom sr=SecureRandom.getInstance("SHA1PRNG");
                byte[] randomB=new byte[64];
                sr.nextBytes(randomB);
                byte[] SID =new byte[1];
                sr.nextBytes(SID);
                byte[] sequenceNumber=new byte[2];
                sr.nextBytes(sequenceNumber);

        mess=request.buildMessage("carolecert.cer","carolekey",randomB,SID,sequenceNumber);
                randomA[0]=(byte)128;
                BigInteger p = new BigInteger(1,randomA);
        BigInteger g = new BigInteger(1,randomB);
        int exp =511;

        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DH");
    DHParameterSpec dhSpec = new DHParameterSpec(p, g, exp);
    keyGen.initialize(dhSpec);
```

```
KeyPair keypair = keyGen.generateKeyPair();

PublicKey DHpublicKey = keypair.getPublic();
PrivateKey DHprivateKey = keypair.getPrivate();
udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);

System.out.println("Receiving U_RUCA.rep ...." );

                mess=new byte[1000];
                udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);
byte[] mess1=new byte[udppacket.getLength()];
System.arraycopy(mess,0,mess1,0,mess1.length);

                ReceiveU_RUCArep receive2=new ReceiveU_RUCArep();
                receivedmess=receive2.getKey("tonycert.cer","carolekey",randomB,mess1);
                X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(receivedmess);
KeyFactory keyFact = KeyFactory.getInstance("DH");
PublicKey publicKey = keyFact.generatePublic(x509KeySpec);

KeyAgreement ka = KeyAgreement.getInstance("DH");
ka.init(DHprivateKey);
ka.doPhase(publicKey, true);
String algorithm = "DES";

// Generate the secret key
SecretKey secretKey1 = ka.generateSecret(algorithm);
System.out.println("Sending U_RUCArep ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCArep request2=new SendU_RUCArep();

    mess=request2.buildMessage("87654321","carolekey","tonycert.cer",SID,sequenceNumber,rando
mA,DHpublicKey);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);

                System.out.println("Receiving U_RUCAconf ...");
                mess=new byte[141];
                udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);

                ReceiveU_RUCAconf receive3=new ReceiveU_RUCAconf();
                receivedmess=receive3.getToken(mess,secretKey1);
                byte[] TokenB=receivedmess;
                System.out.println("Sending U_RUCAconf ...");
                sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_RUCAconf request3=new SendU_RUCAconf();
                mess=request3.buildMessage("87654321",randomB,SID,sequenceNumber,secretKey1);
                udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);
```

```
System.out.println("Setup completed!");
String url= "rtp://127.0.0.1:59150/audio/1";
MediaLocator mrl= new MediaLocator(url);
Processor pros = Manager.createProcessor(mrl);
pros.configure();
while(pros.getState()!=pros.Configured);
pros.setContentDescriptor(null);
TrackControl tr1[] = pros.getTrackControls();
Customg723JavaDecoder codec=new Customg723JavaDecoder(secretKey1);
Codec[] codecs1=new Codec[1];
codecs1[0]=codec;
tr1[0].setCodecChain(codecs1);
pros.realize();
while(pros.getState()!=pros.Realized);
new PlayerWindow(pros);
pros.start();
boolean endcall=false;
byte[] endcallmess=new byte["endcall".getBytes().length];
DatagramPacket packet=new DatagramPacket(endcallmess,endcallmess.length);
DatagramSocket udpsocket = new DatagramSocket(6000);
do
{
        udpsocket.receive(packet);
        if(new String(endcallmess)!="")
        endcall=true;
}
while(endcall==false);
udpsocket.close();
pros.close();
System.out.println("Distante user ended the call!");
System.out.println("Receiving U_ESMSreq ...");
mess=new byte[36];
udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);

ReceiveU_ESMSreq receive4=new ReceiveU_ESMSreq();
receive4.getMessage(mess,secretKey1);
System.out.println("Sending U_ESMSreq ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
sequenceNumber[1]++;
SendU_ESMSreq request4=new SendU_ESMSreq();
mess=request4.buildMessage(SID,sequenceNumber,secretKey1);
udppacket = new DatagramPacket (mess,mess.length,ad,po);
socket.send(udppacket);

System.out.println("Receiving U_ESMSrep ...");
mess=new byte[36];
udppacket = new DatagramPacket (mess,mess.length);
socket.receive(udppacket);

ReceiveU_ESMSrep receive5=new ReceiveU_ESMSrep();
receive5.getMessage(mess,secretKey1);
System.out.println("Sending U_ESMSrep ...");
sequenceNumber[0]++;
if(sequenceNumber[0]==(byte)0)
```

```
	sequenceNumber[1]++;
	SendU_ESMSrep request5=new SendU_ESMSrep();
			mess=request5.buildMessage(SID,sequenceNumber,secretKey1);
			udppacket = new DatagramPacket (mess,mess.length,ad,po);
	socket.send(udppacket);

			System.out.println("Receiving U_ESMSind ...");
			mess=new byte[109];
			udppacket = new DatagramPacket (mess,mess.length);
	socket.receive(udppacket);

			ReceiveU_ESMSind receive6=new ReceiveU_ESMSind();
			receivedmess=receive6.getToken(mess,secretKey1);
			String rec=new String(receivedmess);
			System.out.println("Returned Tokon:"+rec);
			System.out.println("Sending U_ESMSind ...");
			sequenceNumber[0]++;
	if(sequenceNumber[0]==(byte)0)
	sequenceNumber[1]++;
	SendU_ESMSind request6=new SendU_ESMSind();
			mess=request6.buildMessage("12345678",randomA,SID,sequenceNumber,secretKey1);
			udppacket = new DatagramPacket (mess,mess.length,ad,po);
	socket.send(udppacket);

			System.out.println("Receiving U_ESMSconf ...");
			mess=new byte[1000];
			udppacket = new DatagramPacket (mess,mess.length);
	socket.receive(udppacket);
			mess1=new byte[udppacket.getLength()];
	System.arraycopy(mess,0,mess1,0,mess1.length);
			ReceiveU_ESMSconf receive7=new ReceiveU_ESMSconf();
			receivedmess=receive7.getReport("tonycert.cer",mess1,secretKey1);
			FileOutputStream fos=new FileOutputStream("server\\serverreportlog.txt");
			fos.write(receivedmess);
			fos.close();
			rec=new String(receivedmess);
			System.out.println("Distante user report:"+rec);
			System.out.println("Sending U_ESMSconf ...");
			sequenceNumber[0]++;
	if(sequenceNumber[0]==(byte)0)
	sequenceNumber[1]++;
	SendU_ESMSconf request7=new SendU_ESMSconf();
			Date end=new Date();
			String end_time=end.toString();
		mess=request7.buildMessage("87654321","12345678",start_time,end_time,"carolekey",SID,seque
nceNumber,secretKey1);
			udppacket = new DatagramPacket (mess,mess.length,ad,po);
	socket.send(udppacket);
			socket.close();
			System.exit(0);
		}
}
```